INNOVATIVE MINING, PROCESSING, AND APPLICATION OF BIG GRAPHS

A Dissertation
Presented to
The Academic Faculty

By

Yang Zhou

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Computer Science

Georgia Institute of Technology

December 2016

INNOVATIVE MINING, PROCESSING, AND APPLICATION OF BIG GRAPHS

Approved by:

Dr. Ling Liu, Advisor
School of Computer Science
Georgia Institute of Technology

Dr. Shamkant Navathe
School of Computer Science
Georgia Institute of Technology

Dr. Calton Pu
School of Computer Science
Georgia Institute of Technology

Dr. Lakshmish Ramaswamy
SDepartment of Computer Science
University of Georgia

Dr. Jay Lofstead
Scalable System Software Group
Sandia National Laboratories

Date Approved: October 12, 2016

To Mom and Dad

ACKNOWLEDGEMENTS

This dissertation would not have been possible without the generous help and encouragement of many individuals. I would like to express my sincere thanks to everyone who has contributed to the process leading to my dissertation.

First, I wish to thank my advisor and mentor, Professor Ling Liu, for her continuous guidance and encouragement during my PhD study. Her incredible creativity and energy for research has been my biggest source of inspiration during these years. In addition, the collaborations enhanced by her fantastic taste of research problems have helped me shape my flexible research taste and have inspired me to produce novel research outcomes in diverse research fields. I have been extremely fortunate to have worked with her and I hope to be able to pass much of what I learned from her to my students in the future.

Second, I would like to thank my doctoral dissertation committee members: Prof. Shamkant Navathe, Prof. Calton Pu, Prof. Lakshmish Ramaswamy, and Dr. Jay Lofstead, for their invaluable suggestions and comments that have greatly contributed to my thesis. In addition, their welcome suggestions and generous advice have not only helped broaden my visions for my PhD research but also encouraged me to pursue the future academic career.

Third, I have also been fortunate to work as summer research intern at IBM T.J. Watson Research Center and IBM Almaden Research Center. I would like to express my sincere gratitude to my mentors and collaborators, including Dr. Lawrence Chiu, Dr. Chang-Shing Perng, and Dr. Sangeetha Seshadri, for letting me work on real-world research and engineering challenges.

Fourth, I would also like to thank every member of the DiSL Research Group at Georgia Tech for their collaboration and companionship. I convey special thanks to Xianqiang Bao, Kisung Lee, Balaji Palanisamy, Emre Yigitoglu, and Qi Zhang for invaluable discussions and ideas.

Last but not the least, I thank my parents for their unconditional support and constant love. Without the love from my family, this accomplishment would have been impossible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

With continued advances in science and technology, big graph (or network) data, such as World Wide Web, social networks, academic collaboration networks, transportation networks, telecommunication networks, biological networks, and electrical networks, have grown at an astonishing rate in terms of volume, variety, and velocity. Analyzing such big graph data has huge potential to reveal hidden insights and promote innovation in business, science, and engineering domains. However, there exist a number of challenging bottlenecks in developing advanced graph analytics tools in the Big Data era. This dissertation research focus on bridging graph mining and graph processing techniques to alleviate such bottlenecks in terms of both effectiveness and efficiency.

This dissertation had made original contributions on exploring, understanding, and learning big graph data in graph mining, processing and application: First, we have developed a suite of novel graph mining algorithms to analyze real-world heterogeneous information networks. Our algorithmic approaches enable new ways to dive into the correlation structure of big graphs to derive new insights about how heterogeneous entities interact with one another and influence the effectiveness and efficiency of graph clustering, graph classification and graph ranking. Second, we have developed a scalable graph parallel processing framework by exploring parallel processing optimizations at both access tier and computation tier. We have designed a suite of hierarchically composable graph parallel abstractions to enable large-scale graphs to be processed efficiently for iterative graph computation applications. Our approach enables computer hardware resource aware graph partitioning such that parallel graph processing workloads can be well balanced in the presence of highly irregular graph structures and the mismatch of graph access and computation workloads. Third but not the least, we have developed innovative domain specific graph analytics frameworks to understand the hidden patterns in enterprise storage systems and to derive the interesting correlations among various enterprise web services.

These novel graph algorithms and frameworks provide broader and deeper insights for better understanding of tradeoffs in enterprise system design and implementation.

CHAPTER 1
INTRODUCTION

Graph as an expressive data structure is popularly used to model structural relationship between objects in many application domains, such as social networks, web graphs, RDF graphs, sensor networks, protein interaction networks. Heterogeneous information networks are graphs with heterogeneous types of entities and links, associated with static attributes and dynamic and inter-connected activities. Heterogeneous information network analysis has great potential for understanding the ways in which information, ideas, experiences and innovations are spread across information networks.

With continued advances in computing and information technology, heterogeneous information networks have grown at an astonishing rate in terms of volume, variety, and velocity. Mining and processing such content-rich heterogeneous information networks have huge potential to reveal hidden insights and promote innovation in many business, science, and engineering domains. However, the flood of heterogeneous information networks poses great computational challenges in both aspects of algorithm and system: (1) content-rich and heterogeneous graphs with complex structures and attributes; (2) big graphs from multiple arbitrary domains; (3) big graphs with poor quality (e.g., noise and incompleteness); (4) big graphs v.s. limited computational resource; (5) high-degree vertices and skewed vertex degree distribution; and (6) skewed edge weight distribution. Thus, mining and processing heterogeneous information networks with multiple types of links, entities, static attributes and dynamic and inter-connected activities demands for new computational models to address the following new challenges.

- Development of effective big graph mining algorithms that analyze and mine large-scale real heterogeneous information networks.

- Implementation of scalable big graph processing frameworks that speed up the execution of real-world graph applications.
- Development of innovative graph analytics frameworks that perform deep learning and derive new insights in specific research domains, and the design of novel cloud-based big data management and analytics frameworks.

1.1 Technical Challenges

We describe the technical challenges for mining, processing and application of big graph data in more detail as follows.

1.1.1 Data Heterogeneity

As online social media and online shopping sites become ubiquitous, we have witnessed many forms of heterogeneous information networks, such as DBLP bibliography network, Facebook social network and eBay online shopping network, in which entities are of different types, are inter-connected through heterogeneous types of links, representing different kinds of semantic relations, and are associated with multiple types of static attributes and dynamic and inter-connected activities. Heterogeneous information networks analysis has great potential to provide new insights about how information, ideas, experiences and innovations are spread across information networks. However, analyzing and mining such sophisticated heterogeneous networks demands for new computational models and algorithms to address several new challenges: (1) large-scale heterogeneous network analysis often displays features of topological complexity and involves substantial non-trivial computational cost; (2) each type of entities usually associates to one primary social world but participates in many other social worlds, each with domain-specific semantics. It is challenging to efficiently integrate the multiple types of data from multiple information networks into a unified solution space simultaneously; (3) as multiple social networks may be from arbitrary domains, it is challenging to efficiently integrate the multiple types of data

from multiple information networks into a unified distance space simultaneously. Moreover, heterogeneous information network analysis can be more meaningful if it is context aware and only the activity networks that are relevant to the context of interest will be utilized to perform the mining analysis; (4) traditional mining models for networked data are usually based on the existence of vertex homophily, i.e., the principle that similar vertices in nature are connected to each other with links. However, applying vertex homophily alone over heterogeneous information networks is too coarse-grained and may lead to inaccuracy in analysis; and (5) conventional graph mining models are often vertex-centric mining models. Vertex-centric mining and edge-centric mining should be regarded as orthogonal and complimentary dimensions due to their individual mining goals. Relying on either of them alone may result in incomplete and possibly inaccurate analysis results.

1.1.2 Computational Complexity

Many real-world graph applications are iterative graph algorithms, in which vertex or edge weights are updated in each iteration, and these algorithms repeat the iterative processes until convergence. Typical examples include many classical graph algorithms of similarity search, ranking, clustering, classification or collaborative filtering, such as PageRank, EigenTrust, Heat Diffusion Kernel, Random Walk Model, and etc. These graph applications often need to repeatedly self-interact on a single graph or iteratively interact among multiple graphs to discover both direct and indirect relationships between vertices such that the iterative computations involve themselves in lots of matrix-vector multiplications, matrix-matrix multiplications or matrix factorizations. Thus, iterative graph applications are often the compute-intensive applications with high complexity of $O(l * n^2)$ or $O(l * n^3)$ where n is the number of vertices in a graph and l is the number of computation iterations. In addition, iterative graph algorithms usually lead to edge explosion and denser intermediate or result graphs. Existing graph parallel frameworks often fail to perform these compute-intensive applications within acceptable time periods.

1.1.3 System Scalability

Scaling iterative computation on large graphs with billions of vertices and edges is widely recognized as a challenging research problem, which has received heated attention recently. Existing graph processing frameworks often fail to work effectively under the vertex-centric computation model for several scenarios: (1) the algorithms require to load the whole graph into the main memory but the graph and its intermediate computation results together are too big to fit into the available memory; (2) high-degree vertices and their edges combined with the necessary intermediate results are too big to fit into the working memory; (3) the time of computing on a vertex and its edges is much faster than the time to access to the vertex state and its edge data in memory or on disk; (4) the computation workloads on different vertices/edges are significantly imbalanced due to the highly skewed vertex degree/edge weight distribution; (5) the vertex-centric parallel tasks dramatically increase the inter-vertex communication overhead regardless whether the parallel tasks are executed in local memory or shared memory across a cluster of compute nodes; (6) given that different types of iterative graph applications combined with different sizes of graphs often have different resource demands on CPU, memory and disk I/O, a straightforward graph partitioning scheme often can not make good use of limited resource and efficiently respond to the computation requirements under such diverse environments; and (7) lack of flexible graph storage data structure to support utility-aware progressive pruning techniques to further improve the computational performance on big graphs.

1.2 Dissertation Scope and Contributions

To tackle these challenges of big graph analytics, this dissertation research is focused on the development of mining algorithms, processing frameworks and domain-specific applications for effective and scalable big data analytics.

1.2.1 Innovative Mining of Big Graphs

We have developed a suite of innovative graph mining algorithms, including **clustering, classification and ranking of heterogeneous information networks**, to enable new ways to dive into the topology and content of big graphs to derive new insights about how heterogeneous entities interact with each other and influence the effectiveness and efficiency of graph mining. We have made the following original contributions to mining of heterogeneous information networks: (1) devising a unified random walk similarity measure to measure the vertex closeness in terms of both structural and attribute similarities; (2) inventing a social influence-based vertex similarity measure in terms of social influence propagation on both social graph (self-influence) and each of activity graphs (co-influence); (3) designing an edge-centric random walk model to capture both direct and indirect relationships between edges; (4) modeling a heterogeneous information network with multiple types of entities, links, static attributes and dynamic activities in terms of a social graph and multiple associated activity graphs through intra-network or inter-network links; (5) modeling a heterogeneous information network containing multiple types of meta paths in terms of multiple vertex-centric path graphs and multiple edge-centric path graphs; (6) proposing a clustering/classification-based multigraph model to capture the fine-grained clustering/classification-based relationships between pairwise vertices or between pairwise edges about given K clusters/classes; (7) developing reinforcement algorithms to tightly integrate vertex-centric mining and edge-centric mining by mutually enhancing each other; and (8) producing dynamic weight learning methods to efficiently integrate the data from multiple information sources by continuously learning their individual contributions and adjusting their specific weights towards the mining objectives.

1.2.2 Innovative Processing of Big Graphs

We have developed a scalable, efficient, and provably correct two-tier **parallel graph processing framework**, GraphTwist, for executing complex iterative computation tasks over

large-scale graphs on a shared-memory multiprocessor computer. GraphTwist is novel in five aspects: (1) at storage and access tier, modeling a large graph as a compact data structure of 3D cube with source vertex, destination vertex and edge weight as the dimensions, and introducing multi-level hierarchical graph parallel abstraction at different levels of granularity by slice, strip and dice based graph partitioning; (2) devising a dice-based data placement algorithm to store a large graph on disk by minimizing non-sequential disk access and enabling more structured in-memory access, and implementing the index structure, the basic algebra and its core operations to enable fast access to different types of graph partitions; (3) dynamically determining the right level of graph parallel abstraction to maximize sequential access and minimize random access based on the system capacity, the characteristics of graph datasets, and the utility of graph algorithm; (4) designing a regression-based learning method to select the optimal setting for the partitioning parameters, which gives GraphTwist the best performance under the available system resource; and (5) at computation tier, supporting two utility-aware progressive pruning strategies: slice pruning and cut pruning, to further improve the computational performance while preserving the computational utility defined by graph applications, and providing theoretic analysis to quantitatively prove that iterative graph computations powered by utility-aware pruning techniques can achieve a very good approximation with bounds on the introduced error.

1.2.3 Innovative Application of Big Graphs

During past several years, we have developed several domain driven knowledge discovery frameworks to perform deep learning and discover new insights in specific research domains, including storage systems, web services and software engineering. I have collaborated with experts from other communities and developed effective and scalable learning and mining approaches for better analyzing and interpreting various types of data. More specifically, given various types of data from specific domains, we attempt to construc-

t knowledge discovery frameworks with several algorithm and system efforts, including (1) extracting heterogeneous types of entities in specific domains and connecting them to form heterogeneous information networks; (2) understanding those shallow correlations reflected from direct relationship between entities as well as deriving deeper correlations that can only be inferred through reasoning over both direct and indirect correlations in a probabilistic manner; (3) devising innovative mining and learning analytic algorithms to discover interesting and relevant patterns and identify inherent characteristics of the data; and (4) developing innovative parallel or distributed computational frameworks to improve the efficiency and applicability of proposed domain driven knowledge discovery frameworks.

1.3 Dissertation Organization

This dissertation consists of several chapters and each chapter addresses one or more of the challenges described above. In each chapter, we introduce the preliminary concepts and formulates the research problem being addressed, presents the specific solution, followed by extensive experimental results and related work, and conclude the chapter.

In Chapter 2, we present a social influence based clustering framework for analyzing heterogeneous information networks, called SI-CLUSTER, to execute social influence based clustering over heterogeneous information networks by dynamically combining self-influence from social graph and multiple types of co-influence from activity graphs.

In Chapter 3, we propose an activity-edge centric multi-label classification framework for analyzing heterogeneous information networks, AECLASS, to fulfill activity-edge centric multi-label classification of heterogeneous multigraph by integrating structure affinity and label vicinity into a unified classifier with the prior knowledge of multiple activity graphs.

In Chapter 4, we introduce a meta path graph clustering framework, VEPATHCLUSTER, that tightly integrate vertex clustering and edge clustering by mutually enhancing each other with combining different types of meta paths over heterogeneous information network.

In Chapter 5, we present a scalable, efficient, and provably correct two-tier graph processing framework, GRAPHTWIST, in which iterative graph computations powered by utility-aware pruning techniques can achieve a very good approximation with bounds on the introduced error.

In Chapter 6, we model services, attributes, and associated entities in web service libraries, such as providers, consumers, by a heterogeneous service network. A reinforcement algorithm has been provided to tightly integrate ranking and clustering by mutually and simultaneously enhancing each other such that the performance of both can be improved.

In Chapter 7, we apply and extend graph analytics techniques to storage trace analysis for better characterizing important hotspots and understanding hotspot movement patterns.

In Chapter 8, we propose SERVICECLUSTER, a novel heterogeneous SERVICE network CLUSTERing algorithm, for efficient web service analysis.

CHAPTER 2

SI-CLUSTER: SOCIAL INFLUENCE BASED CLUSTERING OF HETEROGENEOUS INFORMATION NETWORKS

Social networks continue to grow in size and the type of information hosted. We witness a growing interest in clustering a social network of people based on both their social relationships and their participations in activity based information networks. In this chapter, we present a social influence based clustering framework for analyzing heterogeneous information networks with three unique features. First, we introduce a novel social influence based vertex similarity metric in terms of both self-influence similarity and co-influence similarity. We compute self-influence and co-influence based similarity in terms of propagating heat diffusion kernel based on social graph and its associated activity graphs and influence graphs respectively. Second, we compute the combined social influence based similarity between each pair of vertices by unifying the self-similarity and multiple co-influence similarity scores through a weight function with an iterative update method. Third, we design an iterative learning algorithm, SI-CLUSTER, to dynamically refine the K clusters by continuously quantifying and adjusting the weights on self-influence similarity and on multiple co-influence similarity scores towards the clustering convergence. To make SI-CLUSTER converge fast, we transformed a sophisticated nonlinear fractional programming problem of multiple weights into a straightforward nonlinear parametric programming problem of single variable. Our experiment results show that SI-CLUSTER not only achieves a better balance between self-influence and co-influence similarities but also scales extremely well for large graph clustering.

2.1 Introduction

Social influence studies the impact of a group of people on an individual member of the group by their opinions or actions. Social influence analysis has great potential for understanding the ways in which information, ideas, experiences and innovations are spread across social networks. As more and more people are engaged in social networks, we witness many forms of heterogeneous social networks in which entities are of different types and are interconnected through heterogeneous types of links, representing different kinds of semantic relations. Analyzing and mining heterogeneous social networks can provide new insights about how people interact with and influence each other and why ideas and opinions on different subjects propagate differently on social networks.

Clustering a heterogeneous social network with multiple types of links, entities, static attributes and dynamic and inter-connected activities demands for new clustering models and distance functions to address the following new challenges.

- The large scale heterogeneous social network analysis often displays features of social complexity and involves substantial non-trivial computational cost. For example, a full version of the DBLP bibliography data contains $964,166$ authors, $6,992$ conferences, $363,352$ keywords and $31,962,786$ heterogeneous links.

- Each type of entities usually associates to one primary social world but participates in many other social worlds, each with domain-specific semantics. How to make good use of the information from various social worlds to provide more informative views of how people influence one another in a given social network? For instance, we may want to utilize the original facebook people network as well as the associated activity networks in the facebook dataset to generate a better clustering of people based on their social influence in terms of both their circle of friends (i.e., self-influence) and their participations in multiple domain specific activity networks (i.e., multiple types of co-influence).

- The information flow between two social worlds may be bidirectional so that we should be careful in differentiating them when we integrate the results from different information networks. For example, Bob may influence his circle of friends (direct or indirect) by his blogs on certain subject and his participation in some tennis tournaments. On the other hand, direct links from a blog (or a tournament) to other blogs (or tournaments) can serve as a recommendation by Bob to its circle of friends.

- As multiple social networks may be from arbitrary domains, it is challenging to efficiently integrate the multiple types of influences from multiple information networks into a unified distance space simultaneously. Moreover, social network clustering can be more meaningful if it is context aware and only the activity networks that are relevant to the context of interest will be utilized to perform the social influence based clustering analysis.

With these new challenges in mind, in this chapter we develop an innovative social influence based graph clustering approach for heterogeneous information networks, SI-CLUSTER. It captures not only the complex attributes of people (vertices) in the social collaboration network but also the nested and complex relationships between people and other types of entities in different information networks in terms of their participations in different activities of interest. Concretely, we categorize the social influence based graph model into three categories: (1) the topological structure of the social network or the activity networks, (2) the single-valued or multi-valued vertex properties that represent relatively stable and static states of vertices in the social network (such as name, sex, age, and multiple education degrees a person may achieve), (3) the nested and complex relationships between the social network and the activity networks (such as multiple activities one may have participated). We show that the social influence based graph clustering for heterogeneous networks demands for a dynamic graph clustering method in contrast to conventional graph clustering algorithms. SI-CLUSTER is designed to cluster large social network with two new criteria: (1) it takes into account both the complex vertex properties and the topo-

logical structure to define the initial influence of a vertex and the weights of its influence propagation to its circle of friends; (2) it computes pairwise vertex closeness by considering not only the social influence patterns (influence-based similarities) based on both direct and indirect social connections existing in the relevant social and activity networks but also the potentially new interactions that have high propagation probabilities based on the existing interactions. A unique characteristics of SI-CLUSTER is its ability of integrating the self-influence and multiple types of co-influences into a unified influence-based similarity measure through iteratively clustering and dynamic weight tuning mechanism.

This chapter makes the following original contributions.

- We integrate different types of links, entities, static attributes and dynamic activities from different networks into a unified influence-based model through the intra-network or inter-network social influences.
- We compute influence-based vertex similarity in terms of heat diffusion based influence propagation on both social graph (self-influence) and each of activity graphs (co-influence).
- A dynamic weight tuning method is provided to combine various influence-based similarities through an iterative learning algorithm, SI-CLUSTER, for social influence based graph clustering. To make the clustering process converge fast, a sophisticated nonlinear fractional programming problem with multiple weights is transformed to a straightforward parametric programming problem of a single variable.
- We perform extensive evaluation on real datasets to demonstrate that SI-CLUSTER can partition the graph into high-quality clusters with cohesive structures and homogeneous social influences.

2.2 Related Work

The most closely related work to this research falls into three areas: social influence analysis, heterogeneous social network analysis and graph clustering. Social influence analysis is gaining attention in recent years. [1] proposed the first provable approximation algorithm for maximizing the spread of influence in a social network. [2] proposed a cascading viral marketing algorithm. [3] proposed a heat-diffusion based viral marketing model with top K most influential nodes. [4] used a user's implicit social graph to generate a friend cluster, given a small seed set of contacts. [5] presented a model in which information can reach a node via the links of the social network or through the influence of external sources.

Recent works on heterogeneous social network analysis [6, 7, 8, 9, 10] combine links and content into heterogeneous information networks to improve the quality of querying, ranking and clustering. [6] proposed a method to model a relational database containing both attributes and links. [7] proposed to learn an optimal linear combination of different relations on heterogeneous social networks in terms of their importance on a certain query. [9] groups objects into pre-specified classes, while generating the ranking information for each type of object in a heterogeneous information network. [10] presented a query-driven discovery system for finding semantically similar substructures in heterogeneous networks.

Graph clustering has attracted active research in the last decade. Most of existing graph clustering techniques have focused on the topological structure based on various criteria, including normalized cuts [11], modularity [12], structural density [13], stochastic flows [14] or clique [15]. K-SNAP [16] and CANAL [17] presented OLAP-style aggregation approaches to summarize large graphs by grouping nodes based on the user-selected attributes. [18] exploited an information-theoretic model for clustering by growing a random seed in a manner that minimizes graph entropy. [19] presented a clustering method which integrates numerical vectors with modularity into a spectral relaxation problem. SA-Cluster [20] and BAGC [21] perform clustering based on both structural and attribute simi-

larities by incorporating attributes as augmented edges to its vertices, transforming attribute similarity to vertex closeness. PathSelClus [22] utilizes limited guidance from users in the form of seeds in some of the clusters and automatically learn the best weights for each meta-path in the clustering process. GenClus [23] proposed a model-based method for clustering heterogeneous networks with different link types and different attribute types.

To our knowledge, this work is the first one to address the problem of social influence based clustering over heterogeneous networks by dynamically combining self-influence from social graph and multiple types of co-influence from activity graphs.

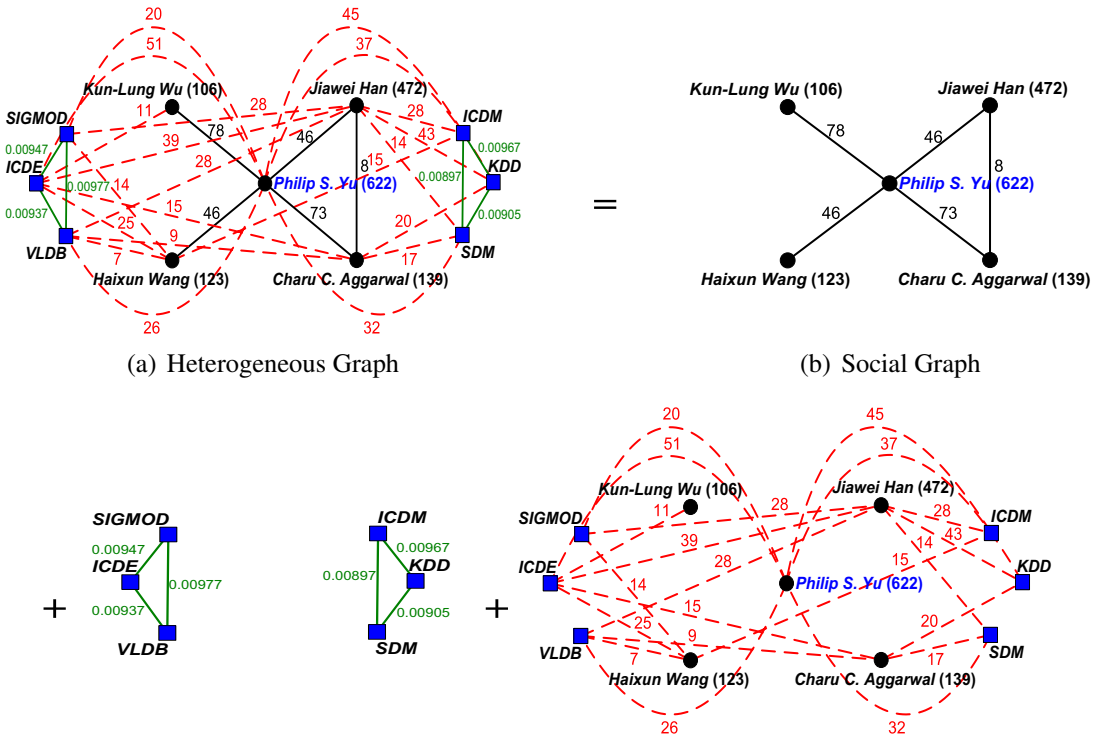
2.3 Problem Statement

We consider three types of information networks in defining a social influence based graph clustering method: (1) the social collaboration network, which is the target of graph clustering and typically a social network of people, such as friend network, co-author network, to name a few; (2) the associated activity networks, such as product purchasing activity network, sport activity network or conference activity network; (3) the influence networks representing bipartite graphs connecting social network and activity networks. We formally define the three types of networks as follows.

A social graph is denoted as $SG = (U, E)$, where U is the set of vertices representing the members of the collaboration network, such as customers or authors, and E is the set of edges denoting the collaborative relationships between members of the collaboration network. We use N_{SG} to represent the size of U, i.e., $N_{SG} = |U|$.

An activity graph is defined by $AG_i = (V_i, S_i)$, where $v \in V_i$ denotes an activity vertex in the i^{th} associated activity network AG_i, and $s \in S_i$ is a weighted edge representing the similarity between two activity vertices, such as functional or manufacture similarity. We denote the size of each activity vertex set as $N_{AG_i} = |V_i|$.

An influence graph is denoted as $IG_i = (U, V_i, S_i, T_i)$, where U, V_i and S_i have the same definitions in the social graph SG and the activity graph AG_i respectively. Every edge



(a) Heterogeneous Graph (b) Social Graph

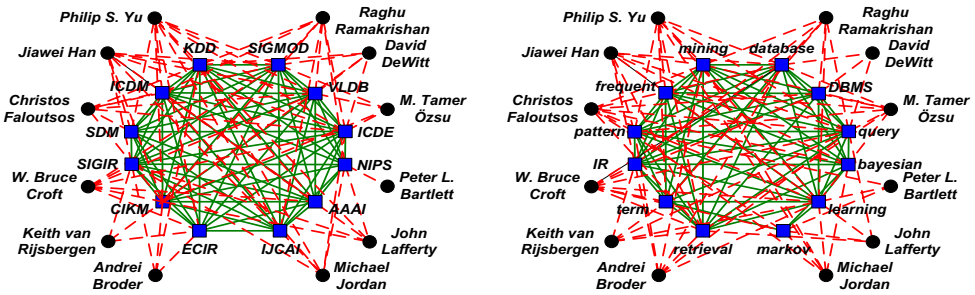
(c) Activity Graph (d) Influence Flow

Figure 2.1: A Heterogeneous Network Example from DBLP

$t \in T_i$, denoted by (u, v), connecting a member vertex $u \in U$ to an activity vertex $v \in V_i$, representing an influence flow between SG and AG_i, such as a purchasing or publishing activity. Thus, IG_i is a bipartite graph.

Given a social graph SG, multiple activity graphs AG_i and various influence graphs IG_i $(1 \leq i \leq N)$, the problem of **Social Influence-based graph Clustering** (SI-Cluster) is to partition the member vertices U into K disjoint clusters U_i, where $U = \bigcup_{i=1}^{K} U_i$ and $U_i \bigcap U_j = \phi$ for $\forall 1 \leq i, j \leq K, i \neq j$, to ensure the clustering results in densely connected groups and each has vertices with similar activity behaviors. A desired clustering result should achieve a good balance between the following two properties: (1) vertices within one cluster should have similar collaborative patterns among themselves and similar interaction patterns with activity networks; (2) vertices in different clusters should have dissimilar collaborative patterns and dissimilar interaction patterns with activities.

Figure 2.1 (a) provides an illustrating example of a heterogeneous information network



(a) Conference Influence Graph (b) Keyword Influence Graph

Figure 2.2: An Illustrating Example of Influence Graphs

extracted from the DBLP dataset. It consists of two types of entities: authors and conferences and three types of links: co-authorship, author-conference, conference similarity. In our SI-Cluster framework, we reorganize a heterogeneous information network into a social graph, multiple activity graphs and multiple influence graphs without loss of information. The heterogeneous network in Figure 2.1 (a) is divided into three subgraphs: a social collaboration graph of authors, a conference activity graph, and an influence graph about author's publishing activity in conferences, as shown in Figures 2.1 (b), (c) and (d), respectively. A red number associated with a red dashed edge quantifies the number of publications that an author published in a conference. A green number on a green edge measures the similarity score between conferences. For ease of presentation, we removed the conference similarities with less than 0.005. A number of mechanisms can be used to compute similarity of conferences. We use RankClus [24] to partition activities into clusters. According to activity's clustering distribution and ranking in each cluster, we calculate the similarities between activities in activity graph. Black numbers in the bracket represent the total amount of publications of an author. Other black numbers on co-author edges denote the number of co-authored papers. A more complex example of influence graph with 12 authors and 12 conferences (or keywords) is presented in Figure 2.2.

2.4 Influence-based Similarity

This section describes how to measure the vertex closeness in terms of self-influence and co-influence models. We first utilize heat diffusion model to capture self-influence based similarity between member vertices in the social graph. Then we use heat diffusion model to construct one co-influence model for each influence graph using a probabilistic classification method to compute co-influence similarities of two vertices in the social graph. Finally, we compute pairwise vertex similarities based on the influence similarity matrix and generate an influence-based pairwise similarity matrix on the social graph for each of its N influence graphs.

2.4.1 Heat Diffusion on Social Graph

Heat diffusion is a physical phenomenon that heat always flows from an object with high temperature to an object with low temperature. In a large social graph SG, experts with many publications often influence other late authors. Consumers purchasing many products may influence other consumers with little purchasing. Thus the spread of influence resembles the heat diffusion phenomenon. Early adopters of a product with many friends or experts on a subject with many coauthors may act as heat sources, transfer their heat to others and diffuse their influence to other majority.

To effectively measure vertex closeness in the social graph in terms of heat diffusion model, we first define the non-propagating heat diffusion kernel on social graph.

Definition 1 *[Non-propagating Heat Diffusion Kernel on Social Graph] Let $SG = (U, E)$ denote a social graph where U is the set of member vertices and E is the edge set denoting the collaborative relationships between members. Let α be the thermal conductivity (the heat diffusion coefficient) of SG. The heat change at vertex $u_i \in U$ between time $t + \Delta t$ and time t is defined by the sum of the heat that it receives from all its neighbors, deducted by*

what it diffuses.

$$\frac{f_i(t+\Delta t)-f_i(t)}{\Delta t}=\alpha\sum_{j:(u_i,u_j)\in E}p_{ij}(f_j(t)-f_i(t)),\ p_{ij}=\begin{cases}\dfrac{n_{ij}}{\sqrt{n_i n_j}},\ (u_i,u_j)\in E_0,\\ 0,\qquad otherwise.\end{cases}\tag{2.1}$$

where $f_i(t)$ is the vertex u_i's temperature at time t. p_{ij} denotes the probability of heat diffusion from u_i to u_j. n_{ij} denotes the weight on edge (u_i, u_j), e.g., the number of co-authored publications, and n_i (or n_j) denotes the amount of heat/influence that u_i (or u_j) has within the social graph, e.g., the number of authored publications. We express the above heat diffusion formulation in a matrix form.

$$\frac{\mathrm{f}(t+\Delta t)-\mathrm{f}(t)}{\Delta t}=\alpha\mathrm{H}\mathrm{f}(t)\tag{2.2}$$

where H *is a $N_{SG}\times N_{SG}$ matrix, called a non-propagating heat diffusion kernel on SG, as the heat diffusion process is defined in terms of one-hop neighbors of heat source.*

$$H_{ij}=\begin{cases}p_{ij}, & (u_i,u_j)\in E,\ i\neq j,\\ -\tau_i, & i=j,\\ 0, & otherwise.\end{cases}\tag{2.3}$$

where $\tau_i=\sum_{(u_i,u_j)\in E, j\neq i}p_{ij}$. τ_i denotes the amount of heat diffused from u_i to all its neighbors.

If we use H to define self-influence similarity between vertices, then the similarity is based on one-hop or direct influence. For those authors who have no joint publications, they are considered to have zero influence on one another, which is unrealistic.

This motivates us to utilize both direct and indirect influence paths between two vertices in computing their vertex similarity. Thus, we define the self-influence similarity using the

propagating heat diffusion kernel, where the heat diffusion process continues until vertices' temperatures converge or the system-defined convergence condition is met. Concretely, by Eq.(2.2), we have the following differential equation when $\Delta t \to 0$.

$$\frac{d\mathrm{f}(t)}{dt} = \alpha \mathrm{H} \mathrm{f}(t) \tag{2.4}$$

Solving this differential equation, we obtain the following Eq.(2.5).

Definition 2 *[Propagating Heat Diffusion Kernel on Social Graph] Let α denote the thermal conductivity,* H *be the non-propagating diffusion kernel of SG and* f(0) *denote an initial heat (influence) column vector at time* 0*, which defines the initial heat distribution on SG. The vertex's thermal capacity at time t, denoted by* f(t)*, is an exponential function with variable t for constant* f(0)*.*

$$\mathrm{f}(t) = e^{\alpha t \mathrm{H}} \mathrm{f}(0) \tag{2.5}$$

We call $e^{\alpha t \mathrm{H}}$ as the propagating heat diffusion kernel. It can be expanded as a Taylor series, where I *is an identity matrix:*

$$e^{\alpha t \mathrm{H}} = \mathrm{I} + \alpha t \mathrm{H} + \frac{\alpha^2 t^2}{2!}\mathrm{H}^2 + \frac{\alpha^3 t^3}{3!}\mathrm{H}^3 + \cdots \tag{2.6}$$

where the heat diffusion reaches convergence, i.e., thermal equilibrium, at time t. Since $e^{\alpha t \mathrm{H}}$ captures both direct and indirect relationships between objects, it reflects the vertex closeness on social graph. We treat it as the self-similarity matrix W_0, i.e., $\mathrm{W}_0 = e^{\alpha t \mathrm{H}}$. Here, the thermal conductivity α is a user specific parameter. We use it as a weight factor for the self-influence similarity in the unified similarity. Figure 2.3 follows the example of Figure 2.1. In Figure 2.3 (a), ochre dashed lines and associated blue numbers represent the self-influence similarity by setting α and t equal to 1.

2.4.2 Heat Diffusion on Influence Graphs

We have presented the use of propagating heat diffusion kernel to measure the self-influence vertex closeness on social graph. In this section we describe how to compute pairwise co-influence similarity for vertices in SG based on one of N associated influence graphs.

Similarly, we first need to define the non-propagating heat kernel on an influence graph. By the definition of influence graph in Section 4.3, we should consider four types of one-hop influence diffusion path in defining the non-propagating heat kernel $\mathrm{H_i}$.

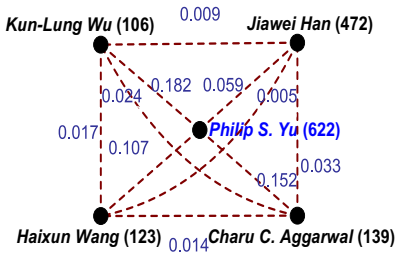
Definition 3 *[Non-propagating Heat Diffusion Kernel on Influence Graphs] We formulate* $\mathrm{H_i}$ *on the influence graph* IG_i *associated to the social graph* SG *and the activity graph* AG_i *by splitting it into four blocks.*

$$\mathrm{H_i} = \begin{bmatrix} \mathrm{A} & \mathrm{B} \\ \mathrm{C} & \mathrm{D} \end{bmatrix} \tag{2.7}$$

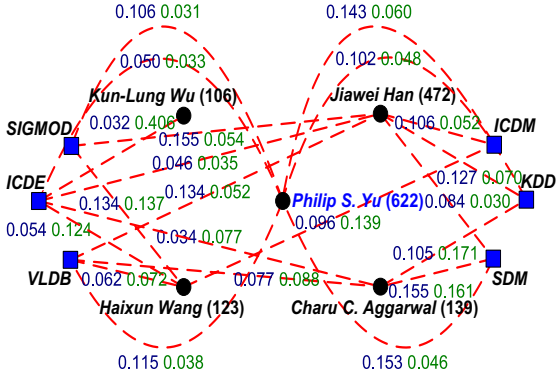
where $\mathrm{B} = [\mathrm{B}_1, \cdots, \mathrm{B}_{\mathrm{N_{AG_i}}}]^T$ *is a* $N_{AG_i} \times N_{SG}$ *matrix representing the social influence of vertices in* AG_i *on members in* SG*, defined by Eq.(2.8);* $\mathrm{C} = [\mathrm{C}_1, \cdots, \mathrm{C}_{\mathrm{N_{SG}}}]^T$ *is a* $N_{SG} \times N_{AG_i}$ *matrix denoting the social influence of members in* SG *on vertices in* AG_i*, defined by Eq.(2.9);* A *is an* $N_{AG_i} \times N_{AG_i}$ *matrix representing the activity similarities, defined by Eq.(2.10); and* D *is a* $N_{SG} \times N_{SG}$ *diagonal matrix.*

$$B_{jk} = \begin{cases} \dfrac{n_{jk}}{\sum_{l=1}^{N_{AG_i}} n_{lk}}, & (u_k, v_j) \in T_i, \\ 0, & otherwise. \end{cases} \tag{2.8}$$

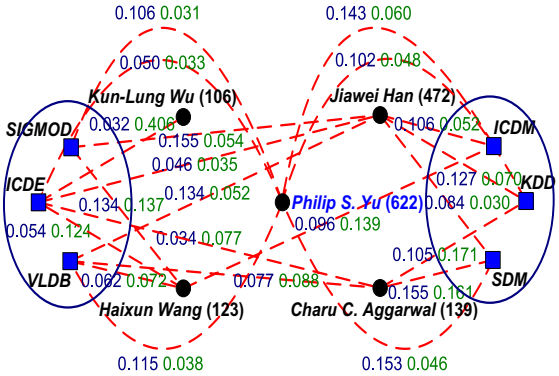
where n_{jk} *is the weight on edge* (u_k, v_j) *and* B_{jk} *computes the influence of* v_j *on* SG *through* u_k *and is defined by* n_{jk} *normalized by the sum of weights on* (u_k, v_l) *for any* v_l *in* AG_i*. For example, the influence of a conference* v_j *on the social graph through an author, say Philip S. Yu, is defined by the number of papers he published in* v_j *normalized by the total number*



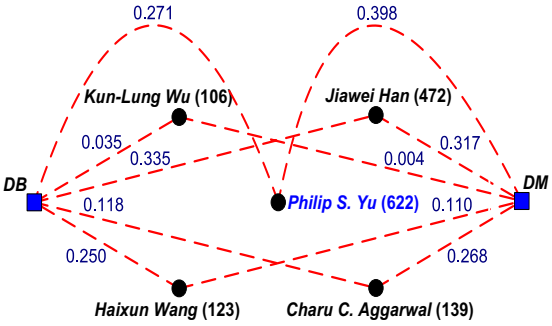
(a) Self-influence Similarity



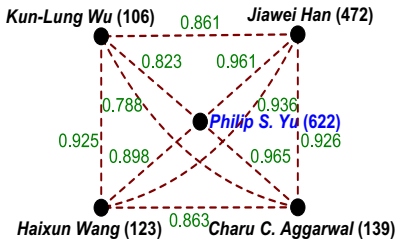
(b) Kernel on Influence Graph



(c) Activity Partition



(d) Influence Propagation



(e) Co-influence Similarity

Figure 2.3: Co-influence Model

of papers authored by him and published in any conference of the conference graph.

$$C_{jk} = \begin{cases} \frac{n_{jk}}{\sum_{l=1}^{N_{SG}} n_{lk}}, & (u_j, v_k) \in T_i, \\ 0, & otherwise. \end{cases} \tag{2.9}$$

where n_{jk} denotes the weight on edge (u_j, v_k) and C_{jk} computes the influence of u_j on AG_i through v_k and is defined by n_{jk} (the amount of papers u_j published in v_k) normalized by the sum of the weights on (u_l, v_k) for any u_l.

$$A_{jk} = \begin{cases} n_{jk}, & (v_j, v_k) \in S_i, \\ -\tau_j, & j = k, \\ 0, & otherwise. \end{cases} \tag{2.10}$$

where n_{jk} represents the similarity between two activity vertices v_j and v_k in the activity graph. $\tau_j = \sum_{(v_j,v_l)\in S_i, l\neq j} A_{jl} + \sum_{(u_l,v_j)\in T_i} B_{jl}$ where τ_j summarizes the influence of activity vertex v_j on other activity vertices and associated member vertices.

In the diagonal matrix D, the diagonal entry D_{jj} in each row is equal to $-\tau_j$ where $\tau_j = \sum_{(u_j,v_l)\in T_i} C_{jl}$. τ_j summarizes the influence of member vertex u_j on all activity vertices.

Definition 4 (Propagating Heat Diffusion Kernel on Influence Graphs) *Let IG_i denote the i^{th} influence graph associated to SG and AG_i, α denote the thermal conductivity,* $\mathrm{H_i}$ *denote the non-propagating diffusion kernel of IG_i and* $\mathrm{f}(0)$ *be an initial heat distribution on IG_i. The vertex's thermal capacity at time t is defined by an exponential function* $\mathrm{f}(t)$ *with variable t for constant* $\mathrm{f}(0)$.

$$\mathrm{f_i}(t) = e^{\alpha t \mathrm{H_i}} \mathrm{f_i}(0) \tag{2.11}$$

where i represents the i^{th} influence graph. $e^{\alpha t \mathrm{H_i}}$ can be expanded as a Taylor series.

$$e^{\alpha t \mathrm{H_i}} = \mathrm{I} + \alpha t \mathrm{H_i} + \frac{\alpha^2 t^2}{2!}\mathrm{H_i^2} + \frac{\alpha^3 t^3}{3!}\mathrm{H_i^3} + \cdots \tag{2.12}$$

where I *is a* $(N_{AG_i} + N_{SG}) \times (N_{AG_i} + N_{SG})$ *identity matrix.*

Figure 2.3 (b) shows the propagating heat diffusion kernel $e^{\alpha t \mathrm{H_{conf}}}$ for the conference influence graph in our running example, where both α and t are set to 1. For presentation clarity, we only show the bidirectional influence flow between authors and conferences with value less than 0.02 in $e^{\alpha t \mathrm{H_{conf}}}$. Associated blue numbers and green numbers quantify the influence flows from author to conference and the influence flows from conference to author respectively.

2.4.3 Co-influence Model

We have defined the propagating heat diffusion kernel $e^{\alpha t \mathrm{H_i}}$ for the influence graph IG_i $(1 \leq i \leq N)$. According to Eq.6.18, in order to conduct heat diffusion on an influence graph and compute pairwise co-influence similarity, we need both $e^{\alpha t \mathrm{H_i}}$ and $\mathrm{f_i}(0)$ on IG_i. $\mathrm{f_i}(0)$ defines the heat sources from which the propagating heat kernel starts its diffusion process.

We observe that the co-influence between a pair of member vertices in the social graph can only be established through their interactions with activity vertices in one of the activity graphs. To make good use of the topological information of AG_i, find good heat sources from AG_i and reduce the commotional cost for large-scale activity graph, we propose to start by partitioning AG_i into M_i disjoint activity clusters, denoted by c_{i1}, c_{i2}, ..., c_{iM_i}. Based on these activity clusters, the initial heat distribution column vector with the size of $(N_{AG_i} + N_{SG}) \times 1$ is defined as follow.

$$\mathrm{f_{ij}}(0) = (p_{ij1}, p_{ij2}, \cdots, p_{ijN_{AG_i}}, 0, 0, \cdots, 0)^T \tag{2.13}$$

where p_{ijk} is the probability of activity vertex v_k belonging to cluster c_{ij} $(1 \leq k \leq N_{AG_i},$

$1 \leq j \leq M_i$). If $p_{ijk} > 0$, then the activity vertex v_k in cluster c_{ij} is chosen as an initial heat source. Note that for each activity vertex v_k, there exists one and only one c_{ij} cluster among the M_i disjoint activity clusters, to which vertex v_k belongs. Thus we have $p_{ijk} = 1$ in $\mathrm{f_{ij}}(0)$. The last N_{SG} entries in $\mathrm{f_{ij}}(0)$ represent the initial heats of member vertices in SG with all 0s. Thus, the initial heat distribution matrix $\mathrm{f_i}(0)$ is defined as $[\mathrm{f_i}(0) = [\mathrm{f_{i1}}(0), \mathrm{f_{i2}}(0), \cdots, \mathrm{f_{iM_i}}(0)]$.

We argue that two members are similar if both of them participate in many activities in the same clusters. We propose a probability based co-influence classification method to classify members into the activity-based clusters and generate the co-influence similarity between members based on the member distribution in each class. We first use $\mathrm{f_{ij}}(0)$ ($1 \leq j \leq M_i$) as the training data and the $e^{\alpha t \mathrm{H_i}}$ as the classifier to execute influence propagation to generate member's probability in each activity-based class. The heat distribution $\mathrm{f_i}(t)$ at time t is then given as follow.

$$\mathrm{f_i}(t) = [\mathrm{f_{i1}}(t), \mathrm{f_{i2}}(t), \cdots, \mathrm{f_{iM_i}}(t)] = e^{\alpha t \mathrm{H_i}}[\mathrm{f_{i1}}(0), \mathrm{f_{i2}}(0), \cdots, \mathrm{f_{iM_i}}(0)] \tag{2.14}$$

Consider conference classes *DM* and *DB* in Figure 2.3 (c), we have the initial conference influence distribution matrix $\mathrm{f_{conf}}(0)$ below.

$$\mathrm{f_{conf}}(0) = [\mathrm{f_{DM}}(0), \mathrm{f_{DB}}(0)] = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}^T \tag{2.15}$$

where 2 columns represent the conference classes *DM* and *DB* and 11 rows represent six conference vertices (*ICDM*, *KDD*, *SDM*, *SIGMOD*, *VLDB* and *ICDE*), and five author vertices (*Philip S. Yu*, *Jiawei Han*, *Charu C. Aggarwal*, *Kun-Lung Wu* and *Haixun Wang*). By Eq.(6.17) with α and time t set to 1, we can generate the final heat distribution vectors $\mathrm{f_{conf}}(t)$ for them, which serve as their influence-based probabilities of belonging to each of

DM and *DB*.

We can further reduce the influence propagation matrix $\mathrm{f_i}(t)$ with the size of $(N_{AG_i} + N_{SG}) \times M_i$ to a $N_{SG} \times M_i$ matrix $\mathrm{f'_i}(t)$ by removing the activity rows without loss of quality. Figure 2.3 (d) shows the influence distribution $\mathrm{f'_{conf}}(t)$ represented by blue numbers in the two different conference classes. The larger the number is, the more influence author has on the conference class.

The pairwise vertex closeness is an important measure of clustering quality. Let $\mathrm{W_i}$ denote the co-influence vertex similarity matrix for influence graph IG_i, M_i be the number of activity classes in IG_i, and $\mathrm{f'_{im}}(t)(j)$ denote the row-wise normalized influence distribution of member $u_j \in U$ on IG_i at time t, i.e., the probability of u_j in the m^{th} class of AG_i. $W_i(j,k)$ representing the co-influence similarity between members u_j and u_k is defined below.

$$\begin{aligned} W_i(j,k) = W_i(k,j) &= 1 - \frac{\sqrt{\sum_{m_i=1}^{M} (\mathrm{f'_{im}}(t)(j) - \mathrm{f'_{im}}(t)(k))^2}}{\sum_{m_i=1}^{M} \mathrm{f'_{im}}(t)(j) + \mathrm{f'_{im}}(t)(k)} \\ &= 1 - \frac{\sqrt{\sum_{m_i=1}^{M} (p_{im(N_{AG_i}+j)} - p_{im(N_{AG_i}+k)})^2}}{\sum_{m_i=1}^{M} p_{im(N_{AG_i}+j)} + p_{im(N_{AG_i}+k)}} \end{aligned} \tag{2.16}$$

The green numbers in Figure 2.3 (e) represents the co-influence based similarity from the conference influence graph.

2.4.4 Unified Influence-based Similarity Measure

The problem of integrating the influence-based similarities on both social graph and multiple influence graphs into a cohesive and unified similarity measure is quite challenging. In this chapter, we propose to use a unified influence-based similarity measure together with an iterative learning algorithm to address this problem.

Let $\mathrm{W_0}$ denote the self-influence similarity from the social graph SG with the weight factor α, $\mathrm{W_i}$ denote the co-influence similarity from the influence graph IG_i $(1 \leq i \leq N)$

with the weight ω_i. The unified similarity function W is defined as follow.

$$\mathrm{W} = \mathrm{W}_0 + \omega_1 \mathrm{W}_1 + \cdots + \omega_N \mathrm{W}_\mathrm{N} \tag{2.17}$$

where $\mathrm{W}_0 = e^{\alpha t \mathrm{H}}$, $\alpha + \sum_{i=1}^{N} \omega_i = N + 1$, $\alpha \geqslant 0$, $\omega_i \geqslant 0$, $i = 1, \cdots, N$.

The unified similarity between any pair of member vertices in SG is defined based on the set of $N + 1$ influence-based similarities.

$$\begin{aligned} d(u_i, u_j) = W(i, j) &= e^{\alpha t H}(i, j) + \omega_1 W_1(i, j) + \cdots + \omega_N W_N(i, j) \\ &= \sum_{k=0}^{\infty} \frac{\alpha^k t^k}{k!} H^k(i, j) + \sum_{k=1}^{N} \omega_k W_k(i, j) \end{aligned} \tag{2.18}$$

2.5 Clustering Algorithm

This section presents our clustering framework, SI-Cluster, that partitions a social graph SG based on both self-influence and co-influence similarities through a unified similarity model among SG, the activity graphs AG_i, and the influence graphs IG_i. SI-Cluster follows the K-Medoids clustering method [25] by using the unified influence-based similarity with the initial weights as an input. At each iteration, we select the most centrally located point in a cluster as a centroid, and assign the rest of points to their closest centroids. The weight update method computes the weighted contributions of each influence-based similarity to both clustering convergence and clustering objective, and updates $N + 1$ weights accordingly after each iteration. This process is repeated until convergence.

2.5.1 Initialization

We will address two main issues in the initialization step: (1) initial weight setup and (2) cluster centroid initialization.

Choosing a weight assignment randomly often results in incorrect clustering results. In

fact, we will prove that there exists one and only one optimal weight assignment to maximize the clustering objective. According to Definition 7 and Theorems 14-17 in Section 8.5.4, we choose parameter $\beta = 0$ and weights $\alpha = \omega_1 = \ldots = \omega_N = 1$ as an initial input. Thus, the dynamic weight update scheme continuously increases weights to important influence-based similarities and decreases weights or assign zero weights to trivial influence-based similarities at each iteration.

Good initial centroids are essential for the success of partitioning clustering algorithms. A member vertex which has a local maximum of the number of neighbors often can diffuse its heat to many vertices along multiple paths. A centroid-based cluster is thus formed when heat is diffused to the margin of the social graph. Thus, we select such K members as the initial centroids $\{c_1^0, ..., c_K^0\}$.

2.5.2 Vertex Assignment and Centroid Update

With K centroids in the t^{th} iteration, we assign each vertex $u_i \in U$ to its closest centroid $c^* = argmax_{c_j^t} d(u_i, c_j^t)$, *i.e.*, a centroid $c^* \in \{c_1^t, ..., c_K^t\}$ with the largest unified similarity from u_i. When all vertices are assigned to some cluster, the centroid will be updated with the most centrally located vertex in each cluster. To find such a vertex, we first compute the "average point" $\overline{u_i}$ of a cluster U_i in terms of the unified similarity matrix as

$$d(\overline{u_i}, u_j) = \frac{1}{|U_i|} \sum_{u_k \in U_i} d(u_k, u_j), \forall u_j \in U_i \tag{2.19}$$

Thus, $d(\overline{u_i}, :)$ is the average unified similarity vector for cluster U_i. Then we find the new centroid c_i^{t+1} in cluster U_i as

$$c_i^{t+1} = argmin_{u_j \in U_i} \|d(u_j, :) - d(\overline{u_i}, :)\| \tag{2.20}$$

Therefore, we find the new centroid c_i^{t+1} in the $(t+1)^{th}$ iteration whose unified similarity vector is the closest to the cluster average.

2.5.3 Clustering Objective Function

The objective of clustering is to maximize intra-cluster similarity and minimize inter-cluster similarity. We first define the inter-cluster similarity.

Definition 5 (Inter-cluster Similarity) *Let $SG = (U, E)$ be the social graph, $W(i, j)$ denote the unified influence-based similarity between u_i and u_j, and U_p and U_q be two clusters of U. The inter-cluster similarity between U_p and U_q is defined as follow.*

$$d(U_p, U_q) = \sum_{u_i \in U_p, u_j \in U_q} d(u_i, u_j) = \sum_{u_i \in U_p, u_j \in U_q} W(i, j) \tag{2.21}$$

This inter-cluster similarity measure is designed to quantitatively measure the extent of similarity between two clusters of U.

Definition 6 *[Graph Clustering Objective Function] Let $SG = (U, E)$ denote a social graph with the weight α and $IG_1, IG_2, \ldots, IG_N$ denote N influence graphs with the weights $\omega_1, \ldots, \omega_N$ where ω_i is the weight for IG_i, and K be a number of clusters. The goal of* SI-Cluster *is to find K partitions $\{U_i\}_{i=1}^K$ such that $U = \bigcup_{i=1}^K U_i$ and $U_i \bigcap U_j = \phi$ for $\forall 1 \leq i, j \leq K, i \neq j$, and the following objective function $O(\{U_l\}_{l=1}^K, \alpha, \omega_1, \ldots, \omega_N)$ is maximized.*

$$\begin{aligned} O(\{U_l\}_{l=1}^K, \alpha, \omega_1, \ldots, \omega_N) &= \frac{\sum_{p=q=1}^K d(U_p, U_q)}{\sum_{p=1}^K \sum_{q=1, q\neq p}^K d(U_p, U_q)} \\ &= \frac{\sum_{p=q=1}^K \sum_{u_i \in U_p, u_j \in U_q} (\sum_{k=0}^{\infty} \frac{\alpha^k t^k}{k!} H^k(i, j) + \sum_{k=1}^N \omega_k W_k(i, j))}{\sum_{p=1}^K \sum_{q=1, q\neq p}^K \sum_{u_i \in U_p, u_j \in U_q} (\sum_{k=0}^{\infty} \frac{\alpha^k t^k}{k!} H^k(i, j) + \sum_{k=1}^N \omega_k W_k(i, j))} \end{aligned} \tag{2.22}$$

subject to $\alpha + \sum_{i=1}^N \omega_i = N + 1$, $\alpha \geqslant 0$, $\omega_i \geqslant 0$, $i = 1, \cdots, N$.

Thus the graph clustering problem can be reduced to three subproblems: (1) cluster assignment, (2) centroid update and (3) weight adjustment, each with the goal of maximizing

the objective function. The first two problems are common to all partitioning clustering algorithms. Thus we focus on the third subproblem, weight adjustment, in the next subsection.

2.5.4 Parameter-based Optimization

The objective function of our clustering algorithm is to maximize intra-cluster similarity and minimize inter-cluster similarity. Theorems 1 and 2 prove that our clustering objective is equivalent to maximize a quotient of two convex functions of multiple variables. It is very hard to perform function trend identification and estimation to determine the existence and uniqueness of solutions. Therefore, we can not directly solve this sophisticated nonlinear fractional programming problem.

Definition 7 *Suppose that* $f(\alpha, \omega_1, \ldots, \omega_N) = \sum_{p=q=1}^{K} \sum_{u_i \in U_p, u_j \in U_q} (\sum_{k=0}^{\infty} \frac{\alpha^k t^k}{k!} H^k(i,j) + \sum_{k=1}^{N} \omega_k W_k(i,j))$ *and* $g(\alpha, \omega_1, \ldots, \omega_N) = \sum_{p=1}^{K} \sum_{q=1, q \neq p}^{K} \sum_{u_i \in U_p, u_j \in U_q} (\sum_{k=0}^{\infty} \frac{\alpha^k t^k}{k!} H^k(i,j) + \sum_{k=1}^{N} \omega_k W_k(i,j))$, *the original clustering goal is rewritten as the following optimization problem (NFPP).*

$$Max\ O(\{U_l\}_{l=1}^{K}, \alpha, \omega_1, \ldots, \omega_N) = \frac{f(\alpha, \omega_1, \ldots, \omega_N)}{g(\alpha, \omega_1, \ldots, \omega_N)} \tag{2.23}$$

subject to $\alpha + \sum_{i=1}^{N} \omega_i = N + 1$, $\alpha \geqslant 0$, $\omega_i \geqslant 0$, $i = 1, \cdots, N$.

Lemma 1 *Let* f *be a function of a single variable on* $\mathbb{R}$. *Then*

(1) f *is concave iff for* $\forall x_1, x_2 \in \mathbb{R}$ *and* $\forall \lambda \in (0,1)$ *we have* $f((1-\lambda)x_1 + \lambda x_2) \geqslant (1-\lambda)f(x_1) + \lambda f(x_2)$.

(2) f *is convex iff for* $\forall x_1, x_2 \in \mathbb{R}$ *and* $\forall \lambda \in (0,1)$ *we have* $f((1-\lambda)x_1 + \lambda x_2) \leqslant (1-\lambda)f(x_1) + \lambda f(x_2)$.

Definition 8 *A set* S *of n-vectors is convex if* $(1-\lambda)x + \lambda x' \in S$ *whenever* $x, x' \in S$, *and* $\lambda \in [0,1]$.

Lemma 2 *Let f be a function of multiple variables with continuous partial derivatives of first and second order on the convex set S and denote the Hessian of f at the point x by $\Pi(x)$. Then*

(1) f is concave iff $\Pi(x)$ is negative semidefinite for $\forall x \in S$.

(2) if $\Pi(x)$ is negative definite for $\forall x \in S$, f is strictly concave.

(3) f is convex iff $\Pi(x)$ is positive semidefinite for $\forall x \in S$.

(4) if $\Pi(x)$ is positive definite for $\forall x \in S$, f is strictly convex.

Lemmas 1, 2 and the detailed proof can be found in [26].

Theorem 1 *$f(\alpha, \omega_1, \ldots, \omega_N)$ is convex on the set $S = \{(\alpha, \omega_1, \ldots, \omega_N) | \alpha + \sum_{i=1}^{N} \omega_i = N + 1, \alpha \geqslant 0, \omega_i \geqslant 0, i = 1, \cdots, N\}$.*

Proof. We first prove that the set S is a convex set. Suppose that two arbitrary $(n+1)$-vectors $x = (\mu_1, \mu_2, \ldots, \mu_{N+1})$ and $x' = (\nu_1, \nu_2, \ldots, \nu_{N+1})$ satisfy the following two constraints: $\sum_{i=1}^{N+1} \mu_i = N + 1$, $\mu_i \geqslant 0$, $\sum_{i=1}^{N+1} \nu_i = N + 1$, $\nu_i \geqslant 0$, $i = 1, \cdots, N + 1$.

For an arbitrary $\lambda \in [0, 1]$, the $(n+1)$-vector $(1-\lambda)x + \lambda x' = ((1-\lambda)\mu_1 + \lambda\nu_1, (1-\lambda)\mu_2 + \lambda\nu_2, \cdots, (1-\lambda)\mu_{N+1} + \lambda\nu_{N+1})$. The sum of each dimension for this $(n+1)$-vector is equal to $(1-\lambda)\sum_{i=1}^{N+1} \mu_i + \lambda \sum_{i=1}^{N+1} \nu_i = (1-\lambda)(N+1) + \lambda(N+1) = N+1$. Thus, $(1-\lambda)x + \lambda x'$ is still in S and S is a convex set.

We then calculate the Hessian matrix of f as follows.

$$\Pi(f)_{ij}(\alpha, \omega_1, \ldots, \omega_N) = D_i D_j f(\alpha, \omega_1, \ldots, \omega_N) \tag{2.24}$$

where D_i is the differentiation operator with respect to the i^{th} argument.

The Hessian becomes $\Pi(f) = [\frac{\partial^2 f}{\partial \alpha^2} \ \frac{\partial^2 f}{\partial \alpha \partial \omega_1} \ \cdots \ \frac{\partial^2 f}{\partial \alpha \partial \omega_N}; \ \frac{\partial^2 f}{\partial \omega_1 \partial \alpha} \ \frac{\partial^2 f}{\partial \omega_1^2} \ \cdots \ \frac{\partial^2 f}{\partial \omega_1 \partial \omega_N}; \ \ldots; \ \frac{\partial^2 f}{\partial \omega_N \partial \alpha} \ \frac{\partial^2 f}{\partial \omega_N \partial \omega_1} \ \ldots \ \frac{\partial^2 f}{\partial \omega_N^2}]$. Since $f(\alpha, \omega_1, \ldots, \omega_N)$ has only one non-linear term $\sum_{p=q=1}^{K} \sum_{u_i \in U_p, u_j \in U_q} \sum_{k=0}^{\infty} \frac{\alpha^k t^k}{k!} H^k(i, j)$, there is one non-zero term $\frac{\partial^2 f}{\partial \alpha^2} = \sum_{p=q=1}^{K} \sum_{u_i \in U_p, u_j \in U_q} \sum_{k=1}^{\infty} k(k-1) \frac{\alpha^{k-2} t^k}{k!} H^k(i, j)$ in the Hessian matrix. We can easily prove that all of its eigenvalues are non-negative. Thus, it is positive-semidefinite for $\forall \alpha$, ω_1, …, $\omega_N \in S$, and $f(\alpha, \omega_1, \ldots, \omega_N)$ is convex on the set S.

Theorem 2 *$g(\alpha, \omega_1, \ldots, \omega_N)$ is convex on S since its Hessian matrix $\Pi(g)$ is positive-semidefinite for $\forall \alpha$, ω_1, …, $\omega_N \in S$.*

The detailed proof is omitted due to space limit. This theorem can be testified by using the above-mentioned similar method.

Theorem 3 *The NFPP problem is equivalent to a polynomial programming problem with polynomial constraints (PPPPC).*

$$Max\ \gamma f(\alpha, \omega_1, \ldots, \omega_N) \tag{2.25}$$

subject to $0 \leqslant \gamma \leqslant 1/g(\alpha, \omega_1, \ldots, \omega_N)$, $\alpha + \sum_{i=1}^{N} \omega_i = N + 1$, $\alpha \geqslant 0$, $\omega_i \geqslant 0$, $i = 1, \cdots, N$.

Proof. If $(\overline{\alpha}, \overline{\omega}_1, \ldots, \overline{\omega}_N, \overline{\gamma})$ is a possible solution of PPPPC, then $\overline{\gamma} = 1/g(\overline{\alpha}, \overline{\omega}_1, \ldots, \overline{\omega}_N)$. Thus $\overline{\gamma} f(\overline{\alpha}, \overline{\omega}_1, \ldots, \overline{\omega}_N) = f(\overline{\alpha}, \overline{\omega}_1, \ldots, \overline{\omega}_N)/g(\overline{\alpha}, \overline{\omega}_1, \ldots, \overline{\omega}_N)$. For any feasible solution $(\alpha, \omega_1, \ldots, \omega_N)$ of NFPP, the constraints of PPPPC are satisfied by setting $\gamma = 1/g(\alpha, \omega_1, \ldots, \omega_N)$, so $\gamma f(\alpha, \omega_1, \ldots, \omega_N) \leqslant \overline{\gamma} f(\overline{\alpha}, \overline{\omega}_1, \ldots, \overline{\omega}_N)$, i.e. $f(\alpha, \omega_1, \ldots, \omega_N)/g(\alpha, \omega_1, \ldots, \omega_N) \leqslant f(\overline{\alpha}, \overline{\omega}_1, \ldots, \overline{\omega}_N)/g(\overline{\alpha}, \overline{\omega}_1, \ldots, \overline{\omega}_N)$.

Conversely, if $(\overline{\alpha}, \overline{\omega}_1, \ldots, \overline{\omega}_N)$ solves NFPP, then for any feasible solution $(\alpha, \omega_1, \ldots, \omega_N, \gamma)$ of PPPPC we have $\gamma f(\alpha, \omega_1, \ldots, \omega_N) \leqslant f(\alpha, \omega_1, \ldots, \omega_N)/g(\alpha, \omega_1, \ldots, \omega_N) \leqslant f(\overline{\alpha}, \overline{\omega}_1, \ldots, \overline{\omega}_N)/g(\overline{\alpha}, \overline{\omega}_1, \ldots, \overline{\omega}_N) = \overline{\gamma} f(\overline{\alpha}, \overline{\omega}_1, \ldots, \overline{\omega}_N)$ with $\overline{\gamma} = 1/g(\overline{\alpha}, \overline{\omega}_1, \ldots, \overline{\omega}_N)$.

Although PPPPC is a polynomial programming problem, the polynomial constraints make it very hard to solve. We further simplify it as an nonlinear parametric programming problem (NPPP).

Theorem 4 *A nonlinear parametric programming problem (NPPP) is defined as $F(\beta) = Max\ \{f(\alpha, \omega_1,$*

$\ldots, \omega_N) - \beta g(\alpha, \omega_1, \ldots, \omega_N)\}$ subject to $\alpha + \sum_{i=1}^{N} \omega_i = N + 1$, $\alpha \geqslant 0$, $\omega_i \geqslant 0$, $i = 1, \cdots, N$. The NFPP problem of Eq.(2.23) is equivalent to this NPPP, i.e., β is a maximum value of NFPP iff $F(\beta) = 0$.

Proof. If $(\overline{\alpha}, \overline{\omega}_1, \ldots, \overline{\omega}_N)$ *is a possible solution of* $F(\beta) = 0$*, then* $f(\overline{\alpha}, \overline{\omega}_1, \ldots, \overline{\omega}_N) - \beta g(\overline{\alpha}, \overline{\omega}_1, \ldots, \overline{\omega}_N) = 0$*. Thus* $f(\alpha, \omega_1, \ldots, \omega_N) - \beta g(\alpha, \omega_1, \ldots, \omega_N) \leqslant f(\overline{\alpha}, \overline{\omega}_1, \ldots, \overline{\omega}_N) - \beta g(\overline{\alpha}, \overline{\omega}_1, \ldots, \overline{\omega}_N) = 0$*. We have* $\beta = f(\overline{\alpha}, \overline{\omega}_1, \ldots, \overline{\omega}_N)/g(\overline{\alpha}, \overline{\omega}_1, \ldots, \overline{\omega}_N) \geqslant f(\alpha, \omega_1, \ldots, \omega_N)/g(\alpha, \omega_1, \ldots, \omega_N)$*. Therefore,* β *is a maximum value of NFPP and* $(\overline{\alpha}, \overline{\omega}_1, \ldots, \overline{\omega}_N)$ *is a feasible solution of NFPP.*

Conversely, if $(\overline{\alpha}, \overline{\omega}_1, \ldots, \overline{\omega}_N)$ *solves NFPP, then we have* $\beta = f(\overline{\alpha}, \overline{\omega}_1, \ldots, \overline{\omega}_N)/g(\overline{\alpha}, \overline{\omega}_1, \ldots, \overline{\omega}_N) \geqslant f(\alpha, \omega_1, \ldots, \omega_N)/g(\alpha, \omega_1, \ldots, \omega_N)$*. Thus* $f(\alpha, \omega_1, \ldots, \omega_N) - \beta g(\alpha, \omega_1, \ldots, \omega_N) \leqslant f(\overline{\alpha}, \overline{\omega}_1, \ldots, \overline{\omega}_N) - \beta g(\overline{\alpha}, \overline{\omega}_1, \ldots, \overline{\omega}_N) = 0$*. We have* $F(\beta) = 0$ *and the maximum is taken at* $(\overline{\alpha}, \overline{\omega}_1, \ldots, \overline{\omega}_N)$*.*

Now we have successfully transformed the original NFPP in Eq.(2.23) into the straightforward NPPP. This transformation can help the algorithm converge in a finite number of iterations. Although it is not clear whether the original objective is concave or convex, the objective $F(\beta)$ of NPPP has the following properties.

Theorem 5 $F(\beta)$ *is a convex function.*

Proof: Suppose that $(\overline{\alpha}, \overline{\omega}_1, \ldots, \overline{\omega}_N)$ *is a possible solution of* $F((1-\lambda)\beta_1 + \lambda\beta_2)$ *with* $\beta_1 \neq \beta_2$ *and* $0 \leqslant \lambda \leqslant 1$*.* $F((1-\lambda)\beta_1 + \lambda\beta_2) = f(\overline{\alpha}, \overline{\omega}_1, \ldots, \overline{\omega}_N) - ((1-\lambda)\beta_1 + \lambda\beta_2) g(\overline{\alpha}, \overline{\omega}_1, \ldots, \overline{\omega}_N) = \lambda(f(\overline{\alpha}, \overline{\omega}_1, \ldots, \overline{\omega}_N) - \beta_2 g(\overline{\alpha}, \overline{\omega}_1, \ldots, \overline{\omega}_N)) + (1-\lambda)(f(\overline{\alpha}, \overline{\omega}_1, \ldots, \overline{\omega}_N) - \beta_1 g(\overline{\alpha}, \overline{\omega}_1, \ldots, \overline{\omega}_N)) \leqslant \lambda \cdot max(f(\overline{\alpha}, \overline{\omega}_1, \ldots, \overline{\omega}_N) - \beta_2 g(\overline{\alpha}, \overline{\omega}_1, \ldots, \overline{\omega}_N)) + (1-\lambda) \cdot max(f(\overline{\alpha}, \overline{\omega}_1, \ldots, \overline{\omega}_N) - \beta_1 g(\overline{\alpha}, \overline{\omega}_1, \ldots, \overline{\omega}_N)) = \lambda F(\beta_2) + (1-\lambda) F(\beta_1)$*. According to Lemma 1, we know that* $F(\beta)$ *is convex.*

Theorem 6 $F(\beta)$ *is a monotonic decreasing function.*

Proof: Suppose that $\beta_1 > \beta_2$ *and* $(\overline{\alpha}, \overline{\omega}_1, \ldots, \overline{\omega}_N)$ *is a possible solution of* $F(\beta_1)$*. Thus,* $F(\beta_1) = f(\overline{\alpha}, \overline{\omega}_1, \ldots, \overline{\omega}_N) - \beta_1 g(\overline{\alpha}, \overline{\omega}_1, \ldots, \overline{\omega}_N) < f(\overline{\alpha}, \overline{\omega}_1, \ldots, \overline{\omega}_N) - \beta_2 g(\overline{\alpha}, \overline{\omega}_1, \ldots, \overline{\omega}_N) \leqslant F(\beta_2)$*.*

Theorem 7 $F(\beta) = 0$ *has a unique solution.*

Proof: Based on the above-mentioned theorems, we know $F(\beta)$ *is continuous as well as decreasing. In addition,* $lim_{\beta \to +\infty} F(\beta) = -\infty$ *and* $lim_{\beta \to -\infty} F(\beta) = +\infty$*.*

2.5.5 Adaptive Weight Adjustment

The procedure of solving this NPPP optimization problem includes two parts: (1) find such a reasonable parameter β ($F(\beta) = 0$), making NPPP equivalent to NFPP; (2) given the parameter β, solve a polynomial programming problem about the original variables. Our weight adjustment mechanism is an iterative procedure to find the solution of $F(\beta) = 0$ and the corresponding weights α, ω_1, ..., ω_N after each iteration of the clustering process. We first generate an initial unified similarity matrix W with equal weights to initialize cluster centroids and partition the social graph. Since $F(\beta)$ is a monotonic decreasing function and $F(0) = Max\{f(\alpha, \omega_1, \ldots, \omega_N)\}$ is obviously non-negative, we start with an initial $\beta = 0$ and solve the subproblem $F(0)$ by using existing fast polynomial programming model to update the weights α, ω_1, ..., ω_N. The updated parameter by $\beta = f(\alpha, \omega_1, \ldots, \omega_N)/g(\alpha, \omega_1, \ldots, \omega_N)$ helps the algorithm enter the next round. The algorithm repeats the above-mentioned iterative procedure until $F(\beta)$ converges to 0.

2.5.6 Clustering Algorithm

By assembling different pieces together, we provide the pseudo code of our clustering algorithm - SI-Cluster in Algorithm 3.

Theorem 8 *The objective function in Algorithm 3 converges to a local maximum in a finite number of iterations.*

Proof. Existing work has studied the convergence properties of the partitioning approach to clustering, such as K-Means [27]. Our clustering follows a similar approach. So the cluster assignment and centroid update steps improve the objective function. In addition, we have explained that nonlinear parametric programming optimization also fast converges a local maximum value. Therefore, the objective function keeps increasing (but $F(\beta)$ keeps decreasing) and converges to a local maximum in a finite number of iterations.

Algorithm 1 Social Influence-based Graph Clustering

Input: a social graph SG, multiple influence graphs IG_i, a cluster number K, initial weights $\alpha = \omega_1 = \ldots = \omega_N = 1$ and a parameter $\beta = 0$.
Output: K clusters $U_1, ..., U_K$.

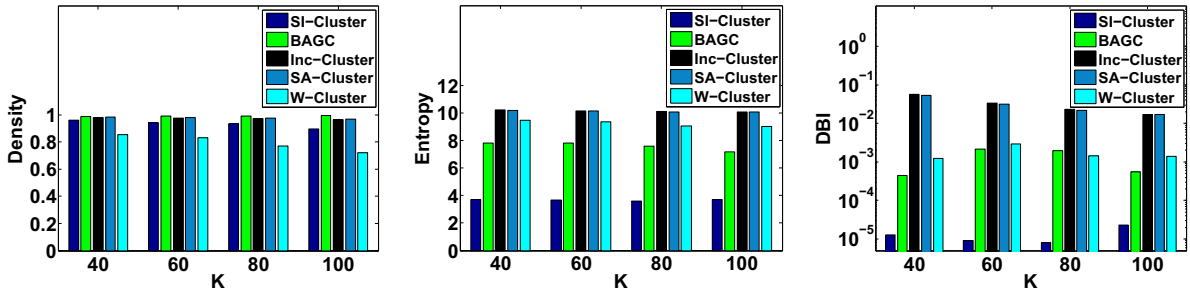
Calculate $W_0, W_1, W_2, \cdots, W_N$, and W;
Select K initial centroids with a local maximum of #neighbors;
Repeat until the objective function $F(\beta)$ converges:
  Assign each vertex u_i to a cluster C^* with a centroid c^* where
  $c^* = argmax_{c_j} d(u_i, c_j)$;
  Update the cluster centroids with the most centrally located point
  in each cluster;
  Solve the NPPP of $F(\beta)$;
  Update $\alpha, \omega_1, ..., \omega_N$;
  Refine $\beta = f(\alpha, \omega_1, \ldots, \omega_N)/g(\alpha, \omega_1, \ldots, \omega_N)$;
  Update W;
Return K clusters $U_1, ..., U_K$.

2.6 Experimental Evaluation

We have performed extensive experiments to evaluate the performance of SI-Cluster on real graph datasets.

2.6.1 Experimental Datasets

We use a full version of the DBLP bibliography data with 964,166 authors (dblp.xml, 836MB, 05/21/2011). We build a social graph where vertices represent authors and edges represent their collaboration relationships, and two associated activity graphs: conference graph and keyword graph. We make use of a multityped clustering framework, RankClus [24], to partition both conferences and keywords into clusters respectively. According to the conference's or keyword's clustering distribution and ranking in each cluster, we calculate the similarities between conferences or keywords. The two associated influence graphs capture how authors in the social graph interact with the activity networks. We also use a smaller DBLP collaboration network with 100,000 highly prolific authors. The third dataset is the Amazon product co-purchasing network with 20,000 products. The two activity networks are product category graph and customer review graph.



(a) density (b) entropy (c) DBI

Figure 2.4: Cluster Quality on Amazon 20,000 Products

2.6.2 Comparison Methods and Evaluation

We compare **SI-Cluster** with three recently developed representative graph clustering algorithms, **BAGC** [21], **SA-Cluster** [20] and **Inc-Cluster** [28], and one baseline clustering algorithm, **W-Cluster**. The last three algorithms integrate entity, link and static attribute information into a unified model. SI-Cluster is our proposed algorithm which incorporates not only links, entities, static attributes but also multiple types of dynamic and interconnected activities into a unified influence-based model. BAGC constructs a Bayesian probabilistic model to capture both structural and attribute aspects. Both SA-Cluster and Inc-Cluster combine both structural and attribute similarities in the clustering decisions by estimating the importance of attributes. W-Cluster combines structural and attribute similarities using the equal weighting factors.

Evaluation Measures We use three measures of to evaluate the quality of clusters $\{U_l\}_{l=1}^{K}$ generated by different methods. The definitions of the metrics are given as follows.

$$density(\{U_l\}_{l=1}^{K}) = \sum_{i=1}^{K} \frac{|\{(u_p, u_q)|u_p, u_q \in U_i, (u_p, u_q) \in E\}|}{|E|} \tag{2.26}$$

$$entropy(\{U_l\}_{l=1}^{K}) = \sum_{i=1}^{N} \frac{\omega_i}{\sum_{p=1}^{N} \omega_p} \sum_{j=1}^{K} \frac{|U_j|}{|U|} entropy(a_i, U_j) \tag{2.27}$$

where ω_i is the weight of influence graph IG_i, $entropy(a_i, U_j) = -\sum_{n=1}^{n_i} p_{ijn} log_2 p_{ijn}$, n_i (or

attribute a_i) is the number of IG_i's activities (or the number of a_i's values) and p_{ijn} is the percentage of vertices in cluster U_j which participate in the n^{th} activity in IG_i (or have value a_{in} on a_i). $entropy(\{U_l\}_{l=1}^{K})$ measures the weighted entropy from all influence graphs (or attributes) over K clusters.

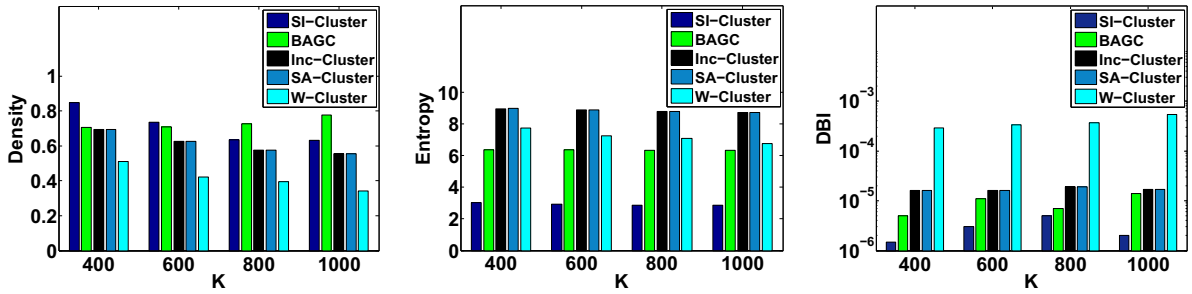
Davies-Bouldin Index (DBI) measures the uniqueness of clusters with respect to the unified similarity measure.

$$DBI(\{U_l\}_{l=1}^{K}) = \frac{1}{K}\sum_{i=1}^{K} max_{j\neq i}\frac{d(c_i, c_j)}{\sigma_i + \sigma_j} \tag{2.28}$$

where c_x is the centroid of U_x, $d(c_i, c_j)$ is the similarity between c_i and c_j, σ_x is the average similarity of vertices in U_x to c_x.

2.6.3 Cluster Quality Evaluation

Figure 2.4 (a) shows the density comparison on Amazon 20, 000 Products by varying the number of clusters $K = 40, 60, 80, 100$. The density values by SI-Cluster, BAGC, Inc-Cluster and SA-Cluster remains 0.89 or higher even when k is increasing. This demonstrates that these methods can find densely connected components. The density values of W-Cluster is relatively lower, in the range of 0.72-0.85 with increasing K, showing that the generated clusters have a very loose intra-cluster structure. Figure 2.4 (b) shows the entropy comparison on Amazon 20, 000 Products with $K = 40, 60, 80, 100$. SI-Cluster has the lowest entropy, while other four algorithms have a much higher entropy than SI-Cluster, since SI-Cluster considers not only static attributes but also multiple types of dynamic and interconnected activities during the clustering process. Other methods can not handle dynamic activities and only treat them as static and isolated attributes. Figures 2.4 (c) shows the DBI comparison on Amazon 20, 000 Products with different K values. SI-Cluster has the lowest DBI of around $0.000008 - 0.000023$, while other methods have a much higher DBI than SI-Cluster. This demonstrates that SI-Cluster can achieve both high intra-cluster similarity and low inter-cluster similarity. This is because SI-Cluster integrates self-influence similarity as



(a) density (b) entropy (c) DBI

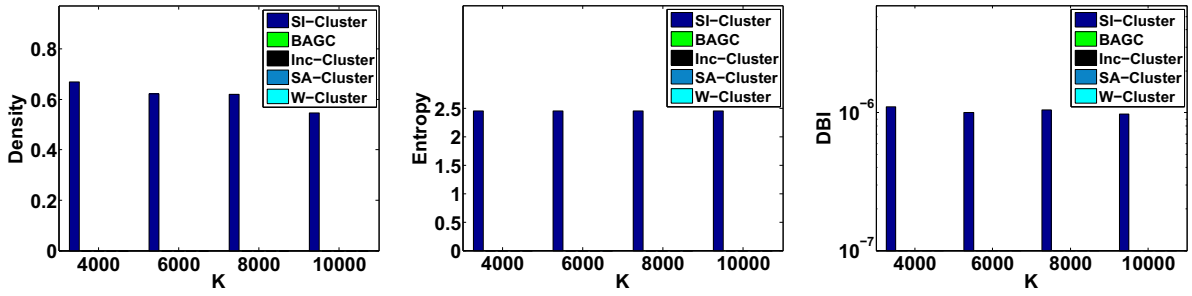
Figure 2.5: Cluster Quality on DBLP 100,000 Authors

well as co-influence similarity with the optimal weights assignment by parameter-based optimization. It fully utilizes the connections between activities and the interactions between members and activities so that the generated clusters have not only similar collaborative patterns but also similar interaction patterns with activities.

Figures 2.5 (a), (b) and (c) show density, entropy and DBI on DBLP with $100,000$ authors when we set $K = 400, 600, 800, 1000$. These three figures have similar trends with Figures 2.4 (a), (b) and (c) respectively. As shown in the figures, SI-Cluster achieves high density values (> 0.63), which is slightly lower than that of BAGC since the probabilistic clustering method partitions vertices into each possible cluster so that the density value by it often increases with K. SI-Cluster achieves a very low entropy around 2.86-3.04, which is obviously better than the other methods (> 6.35). As K increases, the entropy by SI-Cluster remains stable, while the density of SI-Cluster decreases. In addition, SI-Cluster achieves the lowest DBI (< 0.000005) among different methods, while the DBI values by other methods are obviously larger than > 0.000005.

Figures 2.6 (a), (b) and (c) show density, entropy and DBI comparisons on DBLP with $964,166$ authors by varying $K = 4000, 6000,$

$8000, 10000$. Other four methods except SI-Cluster do not work on this large dataset due to the "out of memory" problem with our $8G$ main memory machine. However, SI-Cluster still shows good performance with varying K. It achieves similar high density values ($>$



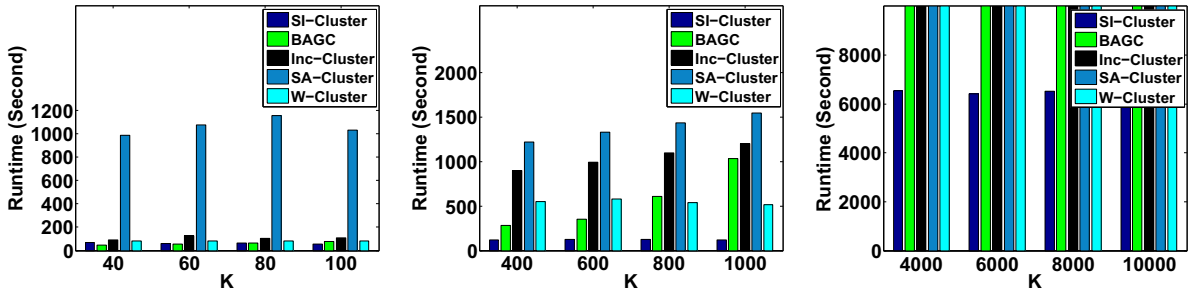
(a) density (b) entropy (c) DBI

Figure 2.6: Cluster Quality on DBLP 964,166 Authors

0.55), much lower entropy of about 2.45, and very low DBI (≈ 0) for different K.

2.6.4 Clustering Efficiency Evaluation

Figures 6.5 (a), (b) and (c) show the clustering time on Amazon 20,000 Products, DBLP $100,000$ and $964,166$ authors respectively. SI-Cluster outperforms all other algorithms in all experiments. When facing with an extremely large dataset, such as DBLP$964,166$, other algorithms cannot work due to the "out of memory" error, while SI-Cluster scales well with large graphs and shows good performance with varying K. We make the following observations on the runtime costs of different methods. First, SA-Cluster is obviously worst than other methods since it needs to perform the repeating random walk distance calculation during each iteration of the clustering process and the distance computation takes more than 80% of the total clustering time. Second, Inc-Cluster, an optimized version of SA-Cluster, is much slower than SI-Cluster, BAGC and W-Cluster since it still needs to incrementally calculate the random walk distance. Third, although W-Cluster compute the random walk distance only once, it still runs on a large scale matrix. Fourth, the performance by BAGC is better than other approaches except SI-Cluster. Although it does not need to repeatedly compute the distance matrix, it needs to iteratively update lots of temporary matrices or interim variables and its computational cost is proportional to K^2 so that it may not work well when facing large K value. In comparison, SI-Cluster reorganizes a



(a) Amazon 20,000 (b) DBLP 100,000 (c) DBLP 964,166

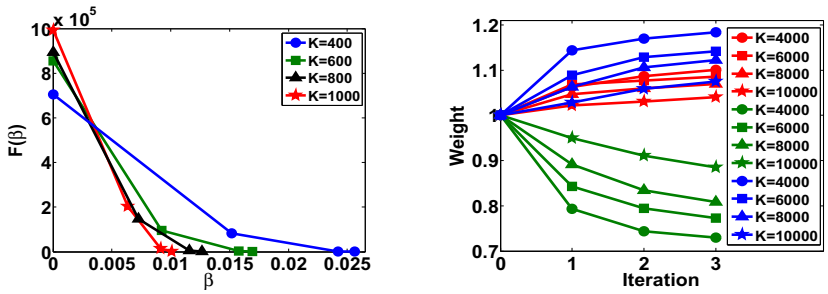
Figure 2.7: Clustering Efficiency

large scale heterogeneous network into multiple small scale subgraphs. It reduces the cost by partitioning activities with the topological information of the activity graph. Furthermore, SI-Cluster calculates influence-based similarity matrices only once. According to Theorems 14-17, solving $F(\beta)$ for a given β is a polynomial programming problem which can be sped up by existing fast polynomial programming model.

2.6.5 Clustering Convergence

Figure 4.13 (a) shows the trend of clustering convergence in terms of the $F(\beta)$ value on DBLP 964, 166 Authors. The $F(\beta)$ value keeps decreasing and has a convex curve when we iteratively perform the tasks of vertex assignment, centroid update and weight adjustment during the clustering process. $F(\beta)$ converges very quickly, usually in three iterations. These are consistent with Theorems 14-17.

Figure 4.13 (b) shows the trend of weight updates on DBLP 964, 166 Authors with different K values: the social graph (red curve), the conference influence graph (green curve) and the keyword influence graph (blue curve). We observe that the graph weights converge as the clustering process converges. An interesting phenomenon is that both the social weight and the keyword weight are increasing but the conference weight is decreasing with more iterations. A reasonable explanation is that people who have many publications in the same conferences may have different research topics but people who have many papers with the same keywords usually have the same research topics, and thus have a higher



(a) $F(\beta)$ (b) Weight Update

Figure 2.8: Clustering Convergence on DBLP 964,166 Authors

collaboration probability as co-authors.

2.6.6 Case Study

We examine some details of the experiment results on DBLP 964, 166 Authors when we set $k = 100$ for both conferences and keywords. Table 4.1 shows author's influence score based on the social influence propagation between authors and keyword partitions. We only present most prolific DBLP experts in the area of data mining or database. When social influence propagation converges, each row represents the influence distribution of an author in each keyword category. We can look upon this influence distribution as a probability based clustering result. On the other hand, each column specifies the influence distribution of different authors in the same keyword category. This influence distribution is considered as a local ranking result.

Table 4.1 actually presents an unbalanced result since the influence propagation process is based on the full DBLP dataset. We know that academic research in the area of database has a longer history and there are more academic conferences or forums focusing on database research. Thus, we choose the same number of top conferences for each research area to better evaluate the quality of our co-influence model. Here, we choose three top conferences from four research areas of database, data mining, information retrieval and artificial intelligence, respectively. The detailed conference list is, DB: VLDB, SIGMOD, ICDE; DM: KDD, ICDM, SDM; IR: SIGIR, CIKM, ECIR; AI: IJCAI, AAAI,

Table 2.1: Influence Scores of Authors Based on Partitions of All Keywords

Author	**Cluster 1 (DB)**	**Cluster 2 (DM)**
Elisa Bertino	0.0568	0.0249
Christos Faloutsos	0.0465	0.0746
Jiawei Han	0.0585	0.0960
Vipin Kumar	0.0146	0.0545
Bing Liu	0.0153	0.0511
David Maier	0.0474	0.0079
Hector Garcia-Molina	0.0603	0.0047
M. Tamer Özsu	0.0408	0.0111
Jian Pei	0.0386	0.0653
Philip S. Yu	0.0606	0.0991

ECAI. Table 4.2 shows author's influence score normalized by conference partitions for each author, i.e., a better probability based clustering result.

2.7 Conclusions

In this chapter, we present a social influence based clustering framework for heterogeneous information networks. First, we integrate different types of links, entities, static attributes and dynamic activities from different networks into a unifying influence-based model. Second, an iterative learning algorithm is proposed to dynamically refine the K clusters by continuously quantifying and adjusting the weights on multiple influence-based similarity scores towards the clustering convergence. Third, we transform a sophisticated nonlinear fractional programming problem of multiple weights into a straightforward nonlinear parametric programming problem of single variable to speed up the clustering process.

Table 2.2: Influence Scores of Authors Based on Partitions of Selected Top Conferences

Author	**AI Cluster**	**DB Cluster**	**DM Cluster**	**IR Cluster**
Elisa Bertino	0.0047	0.7135	0.0055	0.2763
Christos Faloutsos	0.0012	0.4267	0.3950	0.1771
Jiawei Han	0.0883	0.3724	0.3766	0.1628
Vipin Kumar	0.2511	0.1342	0.5198	0.0949
Bing Liu	0.2648	0.1001	0.4004	0.2347
David Maier	0.1570	0.8290	0.0117	0.0023
Hector Garcia-Molina	0.0031	0.8217	0.0075	0.1677
M. Tamer Özsu	0.0017	0.5506	0.1080	0.3397
Jian Pei	0.0876	0.3768	0.3717	0.1639
Philip S. Yu	0.0972	0.3504	0.3763	0.1761

CHAPTER 3

AECLASS: ACTIVITY-EDGE CENTRIC MULTI-LABEL CLASSIFICATION FOR MINING HETEROGENEOUS INFORMATION NETWORKS

Multi-label classification of heterogeneous information networks has received renewed attention in social network analysis. In this chapter, we present an activity-edge centric multi-label classification framework for analyzing heterogeneous information networks with three unique features. First, we model a heterogeneous information network in terms of a collaboration graph and multiple associated activity graphs. We introduce a novel concept of vertex-edge homophily in terms of both vertex labels and edge labels and transform a general collaboration graph into an activity-based collaboration multigraph by augmenting its edges with class labels from each activity graph through activity-based edge classification. Second, we utilize the label vicinity to capture the pairwise vertex closeness based on the labeling on the activity-based collaboration multigraph. We incorporate both the structure affinity and the label vicinity into a unified classifier to speed up the classification convergence. Third, we design an iterative learning algorithm, AEClass, to dynamically refine the classification result by continuously adjusting the weights on different activity-based edge classification schemes from multiple activity graphs, while constantly learning the contribution of the structure affinity and the label vicinity in the unified classifier. Extensive evaluation on real datasets demonstrates that AEClass outperforms existing representative methods in terms of both effectiveness and efficiency.

3.1 Introduction

Multi-label classification has received increasing attention in both data mining and machine learning over the last decade [29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41]. In contrast to single-label classification, multi-label classification analysis adopts a more realistic view

that entities in the real world are often associated with multiple class labels simultaneously. For example, most people in a social network belong to multiple social groups and participate in multiple types of activities with different degrees of engagement. Most of web pages in the web graph may cover multiple topics at different intensities.

Existing multi-label classification efforts for networked data focus on designing effective and yet scalable algorithms [36, 37, 38, 39, 40, 41]. Although previous studies differ from one another in the concrete approaches to mining the linkage structure, to the best of our knowledge, they all suffer from two weaknesses: (1) None of previous studies separate different types of activity graphs from the heterogeneous information networks and exploit the correlations among the set of class labels within each activity graph and across multiple activity graphs; and (2) None of previous works combine both the vertex-centric multi-label classification and the edge-centric multi-label classification to boost the effectiveness and efficiency.

In this chapter we show that by utilizing activity-edge centric approach, we can incorporate the two missing dimensions to improve both the accuracy and the complexity of multi-label classification analysis. First, we argue that entities in the real world may involve themselves in multiple activity networks. These activity networks may provide abundant information about heterogeneous entities and links, and how entities are linked in the context of each of activity networks. We aim to utilize these activity networks to find a natural and cheap way to identify the inter-dependencies among labels. Second, based on different activity networks, an entity can be tagged by a subset of K labels with different class-membership distributions. We model the class-membership distribution for each of activity networks as multi-labeled edges. Third, we consider not only the labels of related vertices but also the possible labels of associated edges to further enhance the accuracy of multi-label classification. We integrate the vertex-centric labeling and edge-centric labeling into a unified classifier with different weights. An iterative method is proposed to learn the weights towards the classification objective.

This chapter makes the following original contributions to multi-label classification for networked data.

- We model a heterogeneous information network in terms of a collaboration graph and multiple associated activity graphs, and cluster activity vertices in each activity graph into K categories with the given K class labels. Clustering each activity graph provides a natural way to capture the dependencies among activity categories within activity graphs.

- We introduce a novel concept of vertex-edge homophily in terms of both vertex labels and edge labels, and transform a general collaboration graph into an activity-based collaboration multigraph by augmenting its edges with class labels from each activity graph through activity-based edge classification.

- We utilize the structure affinity to capture the pairwise topological similarity of vertices and the label vicinity to capture the pairwise vertex closeness based on the labeling on the activity-based collaboration multigraph. We incorporate both the structure affinity and the label vicinity into a unified classifier to speed up the classification convergence.

- We design an iterative learning algorithm, **AEClass**, to dynamically refine the classification result by continuously adjusting the weights on different activity-based edge classification schemes from multiple activity graphs, while constantly learning the contribution of the structure affinity and the label vicinity in the unified classifier. To make the classification process converge fast, a sophisticated nonlinear fractional programming problem with multiple weights is transformed to a straightforward parametric programming problem of a single variable.

- Empirical evaluation over real multi-label datasets demonstrates the competitiveness of AEClass against state-of-the-art methods, in terms of both inference performance and time complexity.

3.2 Related Work

Node classification in networked data has attracted active research in the last decade [42, 43, 44, 45, 46, 47, 9, 48, 49]. LBC [42] is a network-only derivative of the link-based classifier which creates a feature vector for a node by aggregating the labels of neighboring nodes, and then uses logistic regression to build a discriminative model based on these feature vectors. wvRN [43] presented a weighted-vote relational neighbor classifier to solve link-based classification problems based solely on the class labels of linked neighbors. DYCOS [47] exhibited a node classification model in dynamic information networks with both text content and links. RankClass [9] integrates classification and ranking in a mutually enhancing process to provide class summaries for heterogeneous information networks.

Multi-label classification is gaining attention in recent years [29, 30, 31, 32, 33, 34, 35]. Read et al. [31] reduces the complexity and potential for error with a pruning procedure to focus on core relationships within multi-label sets. IBLR [32] proposed a multi-label classification approach to combine model-based and similarity-based inference with the estimation of optimal regression coefficients. LEAD [34] decomposes a multi-label learning task into a set of single-label classification problems with a Bayesian network to encode the conditional dependencies of labels as well as the feature set. Guo and Gu [35] proposed a generalized conditional dependency network for model training using binary classifiers and label predictions using Gibbs sampling inference.

Multi-label classification in networked data has been extensively studied in recent years [36, 37, 38, 39, 40, 41]. Sun et al. [36] presented a hypergraph spectral learning formulation for multi-label classification, where a hypergraph is constructed to exploit the correlation information among different labels. EdgeCluster [37] presented a social-dimension based approach for collective behavior prediction with an edge clustering scheme to extract sparse social dimensions and a linear SVM classifier for discriminative learning. SCRN [40] is a multi-label iterative relational neighbor classifier by considering both network topology

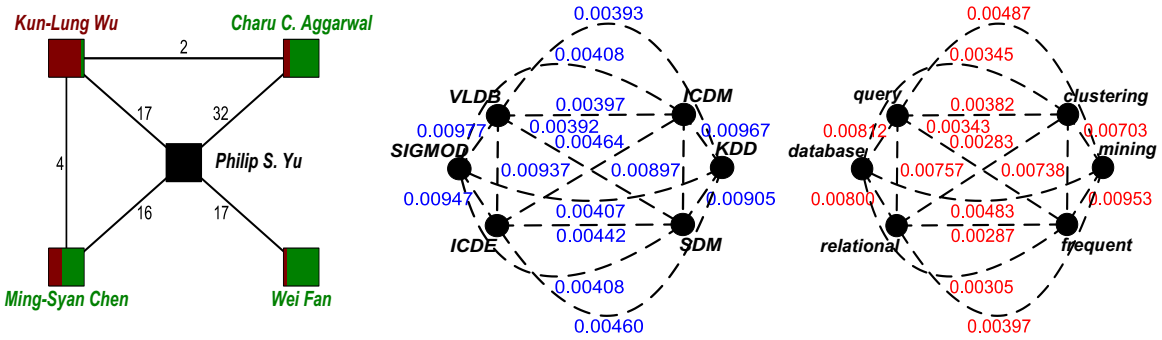
and social context features. PIPL [39] facilitates the multi-label learning process by mining label correlations and instance correlations from the heterogeneous networks.

Recent works on heterogeneous social network analysis [7, 20, 50, 8, 28, 9, 49, 23, 10, 51] combine links and content into heterogeneous information networks to improve the quality of querying, ranking and clustering. Cai et al. [7] proposed to learn an optimal linear combination of different relations on heterogeneous social networks in terms of their importance on a certain query. GenClus [23] proposed a model-based method for clustering heterogeneous networks with different link types and different attribute types. Yu et al. [10] presented a query-driven discovery system for finding semantically similar substructures in heterogeneous networks.

To our knowledge, this work is the first one to address the problem of activity-edge centric multi-label classification of heterogeneous multigraph with the prior knowledge of multiple activity graphs by dynamically adjusting their individual contributions.

3.3 Problem Definition

We address the problem of multi-label classification for networked data by employing our activity-edge centric multi-label classification algorithm. First, we model a heterogeneous information network in terms of two types of information networks: (1) a collaboration graph at the instance level, which is the target of multi-label classification, and (2) a collection of its associated activity graphs at the category level. For example, the DBLP bibliography dataset may consist of three types of vertices: authors, publication venues (e.g., conferences, journals), and title terms in the publications. An author can publish in multiple venues and his papers may contain multiple terms. If the target of multi-label classification is to infer author's labels, then we transform the DBLP dataset into a primary collaboration network for authors at the instance level and two associated activity networks (conference-similarity network and term-similarity network) at the category level. The collaboration network is defined based on both labeled and unlabeled instances with the



(a) Coauthor Graph (b) Conference Graph (c) Term Graph

Figure 3.1: An Illustrating Example from DBLP

given K class labels, where each vertex represents one instance and each edge reflects the collaborative relationships between pairwise instances, e.g., the number of co-authored publications. Each of associated activity networks is constructed with all the associated activities as vertices. Similar activities are linked together with each edge value indicating the similarity between pairwise activities, such as product purchasing activity network, sport activity network or conference activity network. Given that each entity in the collaboration network may participate in multiple activities in each of activity networks, we cluster all activities in each activity networks into K categories. Then we construct a collaboration multigraph by augmenting the original collaboration graph based on N activity networks as follow: For each pair of vertices with an edge in the collaboration graph, if both have participated in at least one of N activity networks, then we will add up to K edges between this pair of vertices.

Figures 5.3 gives an illustrative example extracted from the DBLP dataset, consisting of three graphs: a collaboration graph of authors, a conference activity graph and a term activity graph. For ease of presentation, we only choose the co-authored papers published in three top DB conferences of *SIGMOD*, *VLDB* and *ICDE*, and three top DM conferences of *KDD*, *ICDM* and *SDM*. In Figure 5.3 (a), ochre labels and green labels represent that authors are given predefined class labels of *DB* and *DM* respectively. In addition, ochre block

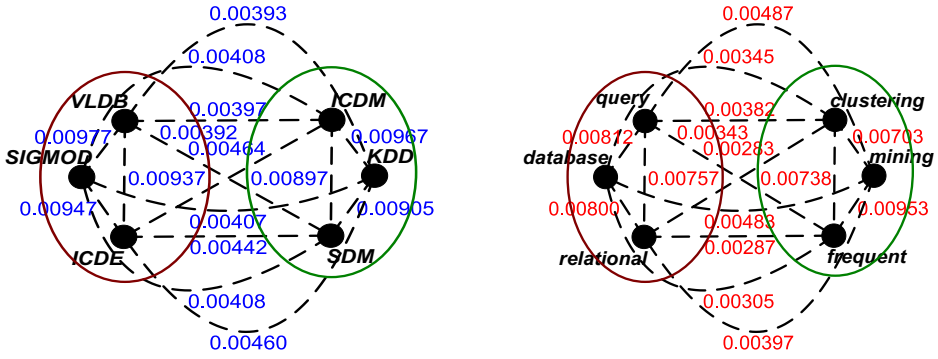
or green block in each vertex rectangle represents the proportion of an author belonging to class *DB* or *DM*. We want to use the label information of four labeled authors to learn the class-membership probabilities of *Philip S. Yu* over classes *DB* and *DM*. In Figure 5.3 (b), blue numbers measure the similarity scores between pairwise conferences. We utilize a multi-typed soft clustering framework, NetClus [50], to cluster conferences and terms into 24 CS research areas [52] simultaneously. According to conference's clustering distribution over 24 categories and ranking score in each category, we calculate the similarities between conferences in the conference activity graph. Similarly, red numbers in Figure 5.3 (c) measure the similarity scores between terms. We then choose a category with the highest probability for each conference or each term as its primary category and put them into the corresponding primary categories. This operation actually produces a hard clustering result for each activity network. As shown in Figure 3.2, the conferences and terms in Figures 5.3 (b) and (c) are put into their individual primary categories respectively.

We formally define the above concepts as follows.

A collaboration graph is denoted as $CG = (V, F)$, where V is the set of vertices representing the entities in CG, such as customers or authors, and F is the set of edges denoting the collaborative relationships between members. We use N_{CG} to represent the size of V, i.e., $N_{CG} = |V|$.

An activity graph is defined by $AG_i = (U_i, F_i)$, where $u \in U_i$ denotes an activity vertex in the i^{th} associated activity network AG_i, and $f \in F_i$ is a weighted edge representing the similarity between two activity vertices, such as functional or manufacture similarity. We denote the size of each activity vertex set as $N_{AG_i} = |U_i|$. The vertex set U_i is partitioned into K disjoint primary categories, denoted by U_{ip} $(1 \leq p \leq K)$, such that $U_i = \bigcup_{p=1}^{K} U_{ip}$ and $U_{ip} \bigcap U_{iq} = \phi$ for $\forall 1 \leq p, q \leq K, p \neq q$ and each activity category U_{ip} is labeled with one of the K class labels, c_p.

Given a collaboration graph $CG = (V, F)$ and its N associated activity graphs $AG_i = (U_i, F_i)$ $(1 \leq i \leq N)$, a collaboration multigraph denoted as $MG = (V, E)$, is an activity-



(a) Conference Clusters (b) Term Clusters

Figure 3.2: Activity Graph Partition

edge augmented multigraph, where V has the same definition in CG and E is the set of edges satisfying the following condition: for each edge $(v_i, v_j) \in F$ in CG, we create a set of parallel edges between the pair of vertices in E. Each set of edges has up to K labeled edges and each edge corresponds to one activity category labeled by c_p ($p \in \{1, \cdots, K\}$) in each of the N activity graphs.

The problem of multi-label classification of multigraph is defined as follows: let $C = \{c_1, c_2, \cdots, c_K\}$ be a finite set of K possible class labels. Given a collaboration multigraph $MG = (V, E)$ with a set of multi-label training instances $V_l \subset V$ initially labeled using the given K class labels, and a set of multi-label testing instances $V_u = V - V_l$ unlabeled. For presentation brevity, we assume that the vertices in V are ordered and the first l vertices are labeled and the remaining vertices are unlabeled. Thus we have $V = \{v_1, \cdots, v_l, v_{l+1}, \cdots, v_{N_{CG}}\}$. Let an instance $v_i \in V$ be associated with a subset of labels in C ,i.e., we use a binary vector $y_i = (y_i^1, y_i^2, \cdots, y_i^K) \in \{0, 1\}^K$, in which $y_i^j = 1$ iff the label c_j is in the label set of v_i. We use $Y = \{y_1, \cdots, y_l, y_{l+1}, \cdots, y_{N_{CG}}\}$ to denote a possible labeling for the instance set V. $Y_l = \{y_1, \cdots, y_l\}$ indicates the observed multi-label set assigned to V_l and $Y_u = Y - Y_l$ represents the multi-label set to be determined. The task of our **activity-edge centric multi-label classification of multigraph** is to use the label information of the training instances in V_l to predict the label set Y_u for the testing instances

in V_u.

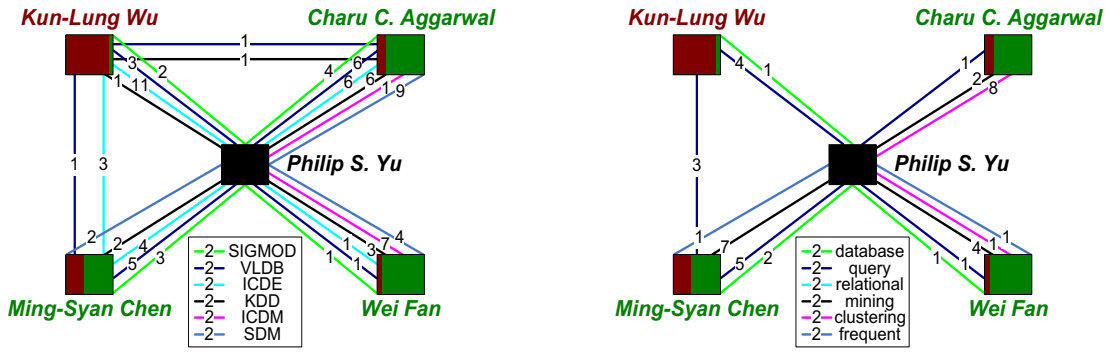
3.4 The AEClass Approach

Compared to existing multi-label relational classifiers outlined in Section 5.1, AEClass improves both the accuracy and the efficiency of multi-label classification by incorporating four mining strategies: (1) activity-based edge classification; (2) edge label dependency; (3) vertex label vicinity; and (4) weight learning. We first introduce the overall design of AEClass. We then describe each part of AEClass in detail in the next subsections.

- Activity-based edge classification, which consists of five tasks. (1) given a collaboration graph CG, choose N suitable activity graphs AG_i based on the specific context defined by the classification objective; (2) cluster all AG_is into K activity categories; (3) construct an label dependency graph based on the clustering of each AG_i to identify inter-dependencies among K class labels; (4) based on K categories of each AG_i, split and classify each unlabeled edge in CG into at most K labeled edges; and (5) transform CG and all AG_is into a unified multigraph MG by integrating N edge classification schemes of CG based on each AG_i weighted by $\omega_1^{(1)}=\cdots=\omega_N^{(1)}=\frac{1}{N}$.

- Activity-edge centric vertex classification, which includes four tasks. (1) initialize a transition probability $\mathbf{T}_j^{(1)}$ of MG; (2) initialize a classification kernel $\mathbf{K}_j^{(1)}$; (3) infer the class-membership vector $\mathbf{X}_j^{(1)}$ on each class c_j; and (4) produce the class-membership vector $\mathbf{Y}_j^{(1)}$ by refining $\mathbf{X}_j^{(1)}$ with label dependency graphs.

- Iterative learning, which has four steps. (1) solve the parametric programming problem for classification objective to update $\alpha^{(t)}, \beta^{(t)}, \omega_1^{(t)}, ..., \omega_N^{(t)}$ ($\alpha^{(t)}, \beta^{(t)}$ are updated if t¿2); (2) adjust the structure affinity $\mathbf{T}_j^{(t+1)}$ of CG with $\omega_1^{(t)}, ..., \omega_N^{(t)}$; (3) update $\mathbf{K}_j^{(t+1)}$ by combining the structure affinity $\mathbf{T}_j^{(t+1)}$ and the label vicinity $(\mathbf{T}_j^{(t+1)}\mathbf{Y}^{(t)})(\mathbf{T}_j^{(t+1)}\mathbf{Y}^{(t)})^T$ weighted by $\alpha^{(t)}$ and $\beta^{(t)}$ ($\alpha^{(t)}=\beta^{(t)}=\frac{1}{2}$ if t=2); and (4) do classification $\mathbf{X}_j^{(t+1)}=\mathbf{K}_j^{(t+1)}\mathbf{X}_j^{(t)}$ and enter the next round.



(a) Conference-based Splitting

(b) Term-based Splitting

Figure 3.3: Edge Splitting

3.4.1 Activity-based Edge Classification

Existing classification models assume the existence of vertex homophily, namely, similar vertices in nature are connected to each other with social links. For example, *Philip S. Yu* and *Wei Fan* have many co-authored works published on *DM* conferences, as shown in Figure 3.4 (a). However, the truth is not always like this. entities that are connected together may be similar in different ways with respect to a given set of K class labels. As is known to all, *Philip S. Yu* and *Ming-Syan Chen* are experts on data mining, i.e., they both have more research publications in the area of data mining than in any other academic area such as database. However, as seen in Figure 3.4 (a), they have more co-authored papers published on *DB* conferences. Thus the vertex homophily is insufficient to accurately infer the possible labels of an author. This motivates us to propose the concept of vertex-edge homophily, the principle that both links and their associated vertices should be similar and likely belong to the same classes, to further improve the accuracy of multi-label classification.

In order to capture the vertex-edge homophily in the multi-label classification, we first perform activity-based edge classification. For each activity graph and the original collaboration graph CG, we first construct an activity-edge augmented collaboration graph CG_i by

examining each edge and the pair of connected entities in CG and splitting each edge into a set of parallel edges based on each activity in AG_i that this pair of entities have in common. The size of each set of parallel edges is at most N_{AG_i}, i.e., the number of activity vertices in AG_i. Figures 3.3 (a) and (b) present the activity-edge augmented collaboration graphs of Figure 5.3 (a) based on conference activities in Figure 5.3 (b) and term activities in Figure (c) respectively. Each edge in Figure 5.3 (a) is divided into multiple edges in terms of the common conference venues or the common title terms in co-authored publications between the pair of co-authors.

However, such activity-based edge augmentation may lead to substantial increase in size of activity-edge augmented collaboration graphs. We address this issue by introducing activity-based edge augmentation with edge classification to efficiently improve the scalability of classification. Concretely, we utilize the clustering result by NetClus, i.e., the probability distribution of each activity over K categories, to infer the class labels of parallel edges in each CG_i over the K categories.

Given the probability of the m^{th} activity in AG_i belonging to cluster (class) c_j produced by NetClus, denoted by $P(L_m = c_j|AG_i)$, we can compute the class-membership probability of edge $(v_p, v_q) \in E$ belonging to class c_j based on AG_i, denoted by $P(L_{pq} = c_j|AG_i)$.

$$P(L_{pq} = c_j|AG_i) = \frac{1}{W(p,q)} \sum_{m=1}^{N_{AG_i}} W^i_m(p,q) P(L_m = c_j|AG_i) \tag{3.1}$$

where $W(p,q)$ represents the value on edge $(v_p, v_q) \in E$ in CG, and $W^i_m(p,q)$ denotes the value on the m^{th} edge between v_p and v_q in CG_i, which is based on the m^{th} activity in AG_i. If there does not exist such an edge between v_p and v_q, then $W^i_m(p,q)$ is equal to 0.

After generating the class-membership distribution of each edge in CG_i, we reduce CG_i to an activity-edge augmented collaboration graph $\overline{CG_i}$ with classified edges by grouping at most N_{AG_i} parallel edges between any pair of vertices in CG_i into at most K parallel edges

in $\overline{CG_i}$.

$$\overline{W}^i_j(p,q) = W(p,q)P(L_{pq} = c_j|AG_i) \tag{3.2}$$

where $\overline{W}^i_j(p,q)$ represents the value on the edge with label c_j between v_p and v_q in $\overline{CG_i}$. For ease of presentation, assuming that *SIGMOD*, *VLDB*, *ICDE*, *database*, *query* and *relational* only belong to class *DB* with the probability of 1, and *KDD*, *ICDM*, *SDM*, *mining*, *clustering* and *frequent* just belong to class *DM* with the probability of 1, two $\overline{CG_i}$s in Figure 3.4 present the edge-classification results of two CG_is in Figures 3.3 respectively.

3.4.2 Activity-edge Centric Vertex Classification

As N edge classification schemes of CG, i.e., $\overline{CG_i}$s ($1 \leq i \leq N$), may have different degree of contributions to vertex classification, we propose to integrate N edge classification schemes into a unified collaboration multigraph with different weighting factors $\omega^{(t)}_1, \cdots, \omega^{(t)}_N$ through dynamic weight tuning mechanism. Thus the unified weight value on the edge with label c_j between v_p and v_q in MG at the t^{th} iteration, denoted by $\mathbf{W}^{(t)}_j(p,q)$, can be computed as follow.

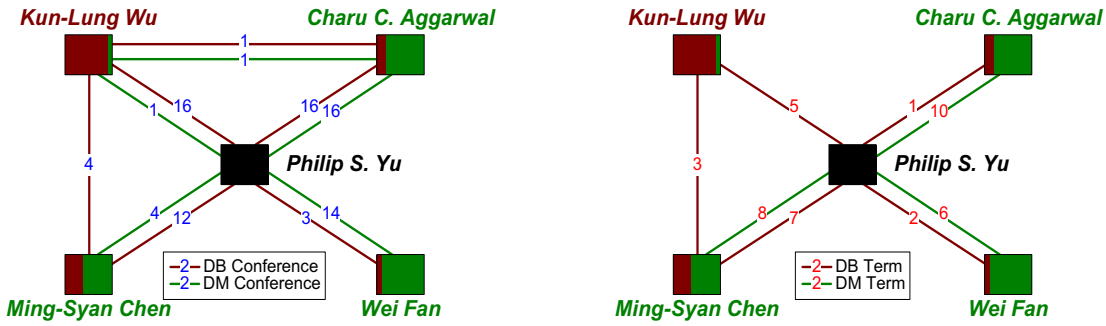
$$\mathbf{W}^{(t)}_j(p,q) = \sum_{i=1}^{N} \omega^{(t)}_i \overline{W}^i_j(p,q) = \sum_{i=1}^{N} \omega^{(t)}_i \sum_{m=1}^{N_{AG_i}} W^i_m(p,q)P(L_m = c_j|AG_i),\ 1 \leq j \leq K \tag{3.3}$$

subject to $\sum_{i=1}^{N} \omega^{(t)}_i = 1$, $\omega^{(t)}_i \geqslant 0$, $i = 1, \cdots, N$.

Note that $\mathbf{W}^{(t)}_j(p,q)$ keeps changing with $\omega^{(t)}_1, \cdots, \omega^{(t)}_N$ through dynamic weight learning. We set the initial $\mathbf{W}^{(1)}_j(p,q)$ with equal weighting factors of $\omega^{(1)}_1, \cdots, \omega^{(1)}_N = \frac{1}{N}$.

Figure 3.5 (a) shows the unified multigraph for our running example in Figure 5.3 by combining the links with the same labels between the same vertex pair from two activity-based edge classification schemes in Figure 3.4 with equal weighting factor of 0.5.

With the unified multigraph MG, we below describe the activity-edge centric vertex



(a) Conference-based Classification (b) Term-based Classification

Figure 3.4: Edge Classification

classification, which integrates the activity-edge labels with the vertex labels among structurally relevant instances through transition probability on collaboration multigraph.

Definition 9 *[Transition Probability on Collaboration Multigraph] Let $MG = (V, E)$ be a collaboration multigraph where V is the set of entity vertices and E is the set of parallel edges denoting the collaborative relationships on different classes between entities of MG. The transition probability on MG at the t^{th} iteration can be defined by normalizing the edge values as follows.*

$$\mathbf{T}_j^{(t)}(p,q) = \begin{cases} \dfrac{\mathbf{W}_j^{(t)}(p,q)}{\sum_{r=1}^{N_{CG}} \sum_{m=1}^{K} \mathbf{W}_m^{(t)}(p,r)}, & p > l, \\ 1, & p = q \leq l, \\ 0, & otherwise. \end{cases} \tag{3.4}$$

where $\mathbf{T}_j^{(t)}(p,q)$ represents the transition probability on the edge with label c_j between v_p and v_q in MG. Here, we assume that Y_l, i.e., the labels of the vertices in V_l, are fixed during the classification process. Figure 3.5 (b) presents the transition probabilities of parallel edges from Philip S. Yu to other authors based on the collaboration multigraph in Figure 3.5 (a).

We express the above transition probability in a matrix form.

$$\mathbf{T}_j^{(t)} = (\mathbf{D}^{(t)})^{-1}\mathbf{W}_j^{(t)} \tag{3.5}$$

where $\mathbf{D}^{(t)}$ *is a diagonal matrix* $\mathbf{D}^{(t)} = diag(1, \cdots, 1, d_{l+1}, \cdots, d_{N_{CG}})$, $1, \cdots, 1$ *specifies* l *ones, and* $d_p = \sum_{r=1}^{N_{CG}} \sum_{m=1}^{K} \mathbf{W}_m^{(t)}(p, r)$ $(l + 1 \leq p \leq N_{CG})$. $\mathbf{T}_j^{(t)}$ *determines the transition probability on those edges with the class label of* c_j *in MG.*

Instead of decomposing the multi-label classification problem into a set of binary classification problems, we construct a unified multi-label classifier by using a single normalizing factor $\mathbf{D}^{(t)}$ *to normalize parallel edges with different class labels. The original transition operation is actually divided into two steps: (1) choose those edges with the objective class label in terms of classification objective; and (2) select an edge with the largest value from the above edges to jump.*

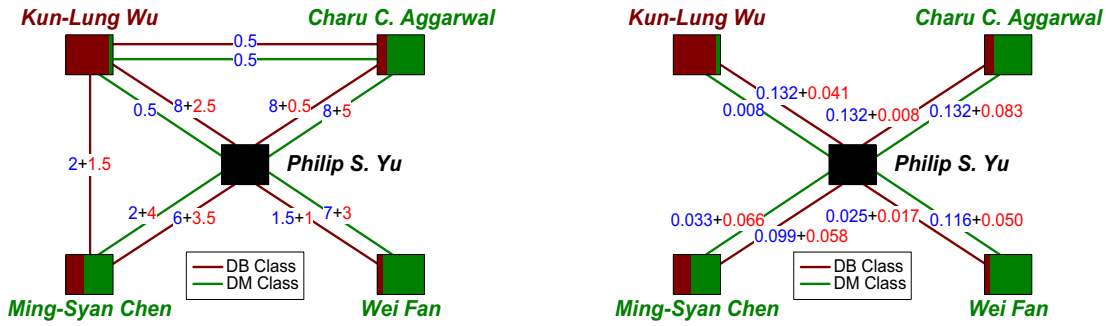
Now we define the initial unified classification kernel $\mathbf{K}_j^{(1)}$*, which only utilizes the structure information of MG, i.e., those edges with label* c_j*, due to the lack of label vicinity at initialization.*

$$\mathbf{K}_j^{(1)} = \mathbf{T}_j^{(1)} \tag{3.6}$$

Since we have ordered the vertices in V *such that the labeled nodes* V_l *are indexed before the unlabeled nodes* V_u*, we rewrite the unified classification kernel* $\mathbf{K}_j^{(1)}$ *as a block matrix.*

$$\mathbf{K}_j^{(1)} = \begin{bmatrix} \mathbf{K}_{jll}^{(1)} & \mathbf{K}_{jlu}^{(1)} \\ \mathbf{K}_{jul}^{(1)} & \mathbf{K}_{juu}^{(1)} \end{bmatrix} = \begin{bmatrix} \mathbf{I} & \mathbf{O} \\ \mathbf{K}_{jul}^{(1)} & \mathbf{K}_{juu}^{(1)} \end{bmatrix} \tag{3.7}$$

where $\mathbf{K}_{jll}^{(1)}$ *is an* $l \times l$ *identity matrix representing the transition probability among labeled vertices, we set the* $l \times (N_{CG} - l)$ *block matrix* $\mathbf{K}_{jlu}^{(1)}$ *to be zero matrix since the labels on the vertices in* V_l *are fixed, the* $(N_{CG} - l) \times l$ *matrix* $\mathbf{K}_{jul}^{(1)}$ *specifies the transition probability from*



(a) Coauthor Multigraph

(b) Transition Probability

Figure 3.5: Multigraph Representation

unlabeled vertices to labeled vertices, and $\mathbf{K}_{juu}^{(1)}$ *is an* $(N_{CG} - l) \times (N_{CG} - l)$ *matrix denoting the transition probability among unlabeled vertices.*

Suppose that the class-membership matrix is denoted by $\mathbf{X} = [\mathbf{X}_1, \mathbf{X}_2, \cdots, \mathbf{X}_K] \in \mathbb{R}^{N_{CG} \times K}$*, for each class-membership vector* $\mathbf{X}_j (1 \leq j \leq K)$ *based on class* c_j*, we use its individual classification kernel* $\mathbf{K}_j^{(t)}$ *to iteratively infer the probabilities of vertices on class* c_j*.*

$$\mathbf{X}_j^{(t)} = \mathbf{K}_j^{(t)} \mathbf{X}_j^{(t-1)} \tag{3.8}$$

Let $\mathbf{X}_j = [\mathbf{X}_{jl}; \mathbf{X}_{ju}]$ be the class-membership vector, where $\mathbf{X}_{jl}$ indicates the probabilities of the labeled vertices in V_l belonging to class c_j, and $\mathbf{X}_{ju}$ represents the probabilities of the unlabeled vertices in V_u belonging to class c_j. Due to the labels on the vertices in V_l are fixed, Eq. (3.8) is equivalent to the following formula.

$$\mathbf{X}_{ju}^{(t)} = \mathbf{K}_{jul}^{(t)} \mathbf{X}_{jl}^{(t-1)} + \mathbf{K}_{juu}^{(t)} \mathbf{X}_{ju}^{(t-1)} \tag{3.9}$$

After the t^{th} iteration, the class-membership matrix is updated as follow.

$$\mathbf{X}^{(t)} = [\mathbf{X}_1^{(t)}, \mathbf{X}_2^{(t)}, \cdots, \mathbf{X}_K^{(t)}] = \begin{bmatrix} \mathbf{X}_{1l} & \mathbf{X}_{2l} & \cdots & \mathbf{X}_{Kl} \\ \mathbf{X}_{1u}^{(t)} & \mathbf{X}_{2u}^{(t)} & \cdots & \mathbf{X}_{Ku}^{(t)} \end{bmatrix} \tag{3.10}$$

Compared to existing multi-label relational classifiers, we argue that AEClass based on the activity-edge augmented collaboration multigraph can significantly improve the performance of multi-label classification: (1) accuracy improvement. Based on the vertex-edge homophily, we classify each edge in *CG* into at most *K* parallel edges in *MG*. During the classification process, AEClass only picks up those vertices and links with the same label as the current objective class c_j, i.e., $\mathbf{K}_j^{(t)}$ and $\mathbf{X}_j^{(t-1)}$, to execute the inference. For example, given the class-membership probabilities of *Ming-Syan Chen* on classes *DB* and *DM* in Figure 3.5 (a), we want to infer the class-membership probabilities of *Kun-Lung Wu* on *DB* and *DM*. AEClass will produce a positive probability on *DB* and a zero probability on *DM* since there exists no edge with label *DM* between these two authors. In contrast, existing classifiers will output a higher probability on *DM* than on *DB* for any positive edge value between two authors in the original *CG* in Figure 5.3 (a). In fact, *Kun-Lung Wu* is known as a database researcher without any data mining publications. (2) efficiency improvement. Based on the vertex homophily, no matter which class the current objective is, existing classifiers need to check each neighbor of a vertex and summarize the labels of all neighbors. In comparison, AEClass performs the similar summary at lower cost. When we classify an edge in *CG* into *m* parallel edges in *MG*, *m* is often much smaller than the number of *K* class labels. Suppose that the current objective class is c_j, for a neighbor of a vertex, there may not exist an edge with label c_j between the vertex and this neighbor. Thus, the number of neighbors of a vertex with edge label c_j can be much smaller than the number of its neighbors, thus reducing the amount of unnecessary computations. Concretely, by utilizing the vertex-edge homophily, AEClass only needs to consider those links with label c_j and associated neighbors and further stops label propagation to the circle of those irrelevant neighbors (without link with label c_j) in the next iterations. For the above example, we can safely ignore the operation of inferring the probability of *Kun-Lung Wu* on *DM* since there exists no edge with label *DM* between two authors. More importantly, AEClass prevents the probability of *Ming-Syan Chen* on *DM* from being diffused to both *Kun-Lung Wu* and

(a) Conference Dependency (b) Term Dependency

Figure 3.6: Edge Label Dependency by Category Similarity

the neighbor-based circle of *Kun-Lung Wu*.

3.4.3 Improvement by Edge Label Dependency

We argue that the underlying correlations among different activity categories can have significant impact on the performance of multi-label classification. Based on activity graph partition, we first define the edge label similarity to capture the inter-dependencies among K activity categories within each of N activity graphs.

Definition 10 (Edge Label Similarity) *Let $AG_i = (U_i, F_i)$ be the i^{th} activity graph ($1 \leq i \leq N$), $S_i(u_m, u_n)$ be the similarity score between two activities $u_m, u_n \in U_i$ in AG_i, and U_{ip} and U_{iq} be two categories of U_i with class labels of $c_p, c_q \in C$ respectively. The activity category similarity between c_p and c_q with respect to AG_i is also referred to as the edge label dependency between two edge labels c_p and c_q, and is defined as follow.*

$$S_i(c_p, c_q) = \begin{cases} \sum_{u_m \in U_{ip}, u_n \in U_{iq}} \frac{S_i(u_m, u_n)}{|U_{ip}| \times |U_{iq}|}, & p \neq q, \\ 1, & p = q. \end{cases} \tag{3.11}$$

Figure 3.6 shows two edge label dependency graph by activity category similarity based on two class labels *DB* and *DM* with respect to the conference graph and the term graph in Figure 3.2 respectively.

We thus incorporate them into our AEClass framework to adjust the class-membership matrix $\mathbf{X}^{(t)}$. The adjusted class-membership vector on class c_j, denoted by $\mathbf{Y}^{(t)}_{ju}$, can be defined by integrating class-membership vectors on other classes in terms of the similarity

scores between class c_j and other classes.

$$\mathbf{Y}^{(t)}_{ju} = \sum_{m=1}^{K} \sum_{i=1}^{N} \omega_i^{(t)} S_i(c_j, c_m) \mathbf{X}^{(t)}_{mu},\ 1 \le j \le K \tag{3.12}$$

where the weight $\omega_i^{(t)}$ for AG_i is the same as in Eq. (3.3). The adjusted class-membership matrix is thus defined as follow.

$$\mathbf{Y}^{(t)} = [\mathbf{Y}^{(t)}_1, \mathbf{Y}^{(t)}_2, \cdots, \mathbf{Y}^{(t)}_K] = \begin{bmatrix} \mathbf{X}_{1l} & \mathbf{X}_{2l} & \cdots & \mathbf{X}_{Kl} \\ \mathbf{Y}^{(t)}_{1u} & \mathbf{Y}^{(t)}_{2u} & \cdots & \mathbf{Y}^{(t)}_{Ku} \end{bmatrix} \tag{3.13}$$

3.4.4 Refinement by Vertex Label Vicinity

One disadvantage of conventional iterative classifiers is that they often need lots of iterations to converge to a stationary distribution and the repeated label propagation causes a non-trivial computational cost. Wang et al. [41] proposed a dynamic label propagation (DLP) model by fusing both data features and data labels to improve the effectiveness on multi-class/multi-label classification. However, the DLP model failed to quantify the weighted contributions from data features and data labels such that it often can not work well on real classification tasks. We model the label vicinity to capture the pairwise vertex closeness based on the labeling on the activity-based collaboration multigraph by following the similar idea. To improve both effectiveness and efficiency of classification, we design an iterative learning method to dynamically refine the classification results by continuously quantifying and adjusting the weights on the structure affinity and on the label vicinity towards the classification objective.

Based on the transition probability $\mathbf{T}^{(t)}_j$ on CG, we define a diffusion process to map the multigraph space into an N_{CG}-dimensional space $\mathbb{R}^{N_{CG}}$, where each element $\phi^{(t)}_j(i) \in \mathbb{R}^{N_{CG}}$ represents the transition probabilities on the edges with label c_j from vertex v_i to the other vertices and $P(\phi^{(t)}_j(i)) = \mathcal{N}(\phi^{(t)}_j(i)|\mathbf{T}^{(t)}_j)$. On the other hand, based on a heuristics rule: two instance vertices with highly similar class-membership distributions are likely to be

highly similar to each other in the input multigraph space, $\mathbf{Y}^{(t)}(\mathbf{Y}^{(t)})^T$ can be viewed as the similarity between vertices based on the class-membership distribution. Similarly, we map this label-based similarity space into an N_{CG}-dimensional space $\mathbb{S}^{N_{CG}}$, where each entry $\varphi^{(t)}(i) \in \mathbb{S}^{N_{CG}}$ specifies the label-based similarity between vertex v_i and the other vertices and $P(\varphi^{(t)}(i)) = \mathcal{N}(\varphi^{(t)}(i)|\mathbf{Y}^{(t)}(\mathbf{Y}^{(t)})^T)$. We then define a linear projection operation based on $\mathbf{T}_j^{(t+1)}$.

$$\phi_j^{(t+1)} = \mathbf{T}_j^{(t+1)}\varphi^{(t)} \tag{3.14}$$

where $\phi_j^{(t+1)} = [\phi_j^{(t+1)}(1); \phi_j^{(t+1)}(2); \cdots; \phi_j^{(t+1)}(N_{CG})]$ and $\varphi^{(t)} = [\varphi^{(t)}(1);$ $\varphi^{(t)}(2); \cdots; \varphi^{(t)}(N_{CG})]$.

With the linear projection, we generate the following formula.

$$P(\phi_j^{(t+1)}|\varphi^{(t)}) = \mathcal{N}(\phi_j^{(t+1)}|\mathbf{T}_j^{(t+1)}\varphi^{(t)}) \tag{3.15}$$

The corresponding marginal distribution is given below.

$$\begin{aligned} P(\phi_j^{(t+1)}) &= \int \mathcal{N}(\varphi^{(t)}|\mathbf{Y}^{(t)}(\mathbf{Y}^{(t)})^T)\mathcal{N}(\phi_j^{(t+1)}|\mathbf{T}_j^{(t+1)}\varphi^{(t)})d\varphi^{(t)} \\ &= \mathcal{N}(\phi_j^{(t+1)}|\mathbf{T}_j^{(t+1)}\mathbf{Y}^{(t)}(\mathbf{Y}^{(t)})^T(\mathbf{T}_j^{(t+1)})^T) \end{aligned} \tag{3.16}$$

Since directly combining $\varphi^{(t)}(\varphi^{(t)})^T$ into the unified classification kernel $\mathbf{K}_j^{(t+1)}$ may lead to a degeneration at the beginning of classification if the learned label information of vertices in V_u is not enough to infer the label-based similarity scores, we adjust $\mathbf{K}_j^{(t+1)}$ by

integrating the label-based similarity through $\mathbf{T}_j^{(t)}$.

$$\mathbf{K}_j^{(t+1)} = \alpha^{(t+1)}\mathbf{T}_j^{(t+1)} + \beta^{(t+1)}(\mathbf{T}_j^{(t+1)}\mathbf{Y}^{(t)})(\mathbf{T}_j^{(t+1)}\mathbf{Y}^{(t)})^T \tag{3.17}$$

subject to $\alpha^{(t+1)} + \beta^{(t+1)} = 1, \alpha^{(t+1)}, \beta^{(t+1)} \geq 0$.

$\alpha^{(t+1)}$ and $\beta^{(t+1)}$ are weighting factors to balance two kinds of similarity scores. The label vicinity $(\mathbf{T}_j^{(t+1)}\mathbf{Y}^{(t)})(\mathbf{T}_j^{(t+1)}\mathbf{Y}^{(t)})^T$ quantitatively measures the extent of similarity between vertices and their neighbors based on the current labeling.

3.4.5 Weight Learning

Classification analysis often utilizes the F1 score, i.e., the harmonic mean of precision and recall, to evaluate the accuracy of testing instances. The objective of multi-label classification of multigraph is to maximize the Macro-F1 score [53], i.e., the unweighted mean of F1 score on classes. To define the Macro-F1 score, we first introduce an indicator function.

$$\mathcal{I}(\hat{y}_i^j = 1) = \begin{cases} 1, & \hat{y}_i^j = 1, \\ 0, & \hat{y}_i^j = 0. \end{cases} \tag{3.18}$$

where $\mathcal{I}(\hat{y}_i^j = 1)$ indicates whether the label c_j is assigned to an instance vertex v_i.

Definition 11 *[Macro-F1] Let $MG = (V, E)$ be a collaboration multigraph, y_i be the true label vector of the i^{th} instance vertices in V and $\hat{y}_i$ be the predicted label vector, and the Macro-F1 score is defined below.*

$$Macro{-}F1 = \frac{1}{K}\sum_{j=1}^{K}\frac{2\sum_{i=l+1}^{N_{CG}}\mathcal{I}(\hat{y}_i^j = 1)y_i^j}{\sum_{i=l+1}^{N_{CG}}\mathcal{I}(\hat{y}_i^j = 1) + \sum_{i=l+1}^{N_{CG}} y_i^j} \tag{3.19}$$

Assuming $\theta = \max_{i,j}\{\mathbf{Y}(i, j) : l + 1 \leq i \leq N_{CG}, 1 \leq j \leq K\}$, we define an s-shape function

to approximate the indicator function.

$$\mathcal{S}(\mathbf{Y}(i,j)) = \begin{cases} 1, & \mathbf{Y}(i,j) = \theta, \\ 0, & \mathbf{Y}(i,j) = 0, \\ \mathbf{Y}(i,j)/\theta, & otherwise. \end{cases} \tag{3.20}$$

The decision rule determining $\hat{y}_i^j = 1$ if $\mathbf{Y}(i,j) > \theta/2$, i.e., $\mathcal{S}(\mathbf{Y}(i,j)) > 0.5$ is represented as follow.

$$\mathcal{I}(\hat{y}_i^j = 1) = \mathcal{I}(\mathbf{Y}(i,j) > \theta/2) = \mathcal{I}(\mathcal{S}(\mathbf{Y}(i,j)) > 0.5) \approx \mathcal{S}(\mathbf{Y}(i,j)) \tag{3.21}$$

The Macro-F1 score is thus approximated as follow.

$$Macro{-}F1 \approx \frac{1}{K}\sum_{j=1}^{K} \frac{2\sum_{i=l+1}^{N_{CG}} \mathbf{Y}(i,j) y_i^j}{\sum_{i=l+1}^{N_{CG}} \mathbf{Y}(i,j) + \sum_{i=l+1}^{N_{CG}} \theta y_i^j} \tag{3.22}$$

According to Eqs.(3.3)-(3.17), the Macro-F1 score is a fractional function of multi variables $\alpha, \beta, \omega_1, \cdots, \omega_N$ with non-negative real coefficients. On the other hand, the numerator and the denominator of Macro-F1 are both polynomial functions of the above variables. Without loss of generality, we rewrite Eq.(3.22) as follow.

$$\begin{gathered} Macro{-}F1 \approx \frac{\sum_{i=1}^{m} a_i(\alpha)^{b_i}(\beta)^{c_i} \prod_{j=1}^{N}(\omega_j)^{d_{ij}}}{\sum_{i=1}^{n} o_i(\alpha)^{p_i}(\beta)^{q_i} \prod_{j=1}^{N}(\omega_j)^{r_{ij}}}, \\ a_i, b_i, c_i, d_{ij}, o_i, p_i, q_i, r_{ij} \geq 0, b_i, c_i, d_{ij}, p_i, q_i, r_{ij} \in \mathbb{Z} \end{gathered} \tag{3.23}$$

where there are m polynomial terms in the numerator and n polynomial terms in the denominator, a_i and o_i are the coefficients of the i^{th} terms respectively, and $b_i, c_i, d_{ij}, p_i, q_i, r_{ij}$ are the exponents of corresponding variables in the i^{th} terms respectively.

Definition 12 *[Multigraph Classification Objective] Let $MG = (V, E)$ be a collaboration multigraph, $\alpha, \beta, \omega_1, \cdots, \omega_N$ are the weighting factors defined in Eqs.(3.3) and (3.17), respectively. The goal of multi-label classification of multigraph is to maximize the Macro-F1 score.*

$$\max_{\alpha,\beta,\omega_j} Macro{-}F1 \approx \max_{\alpha,\beta,\omega_j} \frac{\sum_{i=1}^{m} a_i(\alpha)^{b_i}(\beta)^{c_i} \prod_{j=1}^{N}(\omega_j)^{d_{ij}}}{\sum_{i=1}^{n} o_i(\alpha)^{p_i}(\beta)^{q_i} \prod_{j=1}^{N}(\omega_j)^{r_{ij}}} \tag{3.24}$$

subject to $\alpha + \beta = 1$, $\alpha, \beta \geqslant 0$, $\sum_{j=1}^{N} \omega_j = 1$, $\omega_j \geqslant 0$, $j = 1, \cdots, N$.

For ease of presentation, we revise the original objective as the following nonlinear fractional programming problem (NFPP).

Definition 13 *[Nonlinear Fractional Programming Problem] Let* $f(\alpha, \beta, \omega_1, \ldots, \omega_N) = \sum_{i=1}^{m} a_i(\alpha)^{b_i}(\beta)^{c_i} \prod_{j=1}^{N}(\omega_j)^{d_{ij}}$ *and* $g(\alpha, \beta, \omega_1, \ldots, \omega_N) = \sum_{i=1}^{n} o_i(\alpha)^{p_i}(\beta)^{q_i} \prod_{j=1}^{N}(\omega_j)^{r_{ij}}$*, the classification goal is revised below.*

$$\max_{\alpha,\beta,\omega_1,\ldots,\omega_N} \frac{f(\alpha, \beta, \omega_1, \ldots, \omega_N)}{g(\alpha, \beta, \omega_1, \ldots, \omega_N)} \tag{3.25}$$

subject to $\alpha + \beta = 1$, $\alpha, \beta \geqslant 0$, $\sum_{i=1}^{N} \omega_i = 1$, $\omega_i \geqslant 0$, $i = 1, \cdots, N$.

Our classification objective is equivalent to maximize a quotient of two polynomial functions of multiple variables. It is very hard to perform function trend identification and estimation to determine the existence and uniqueness of solutions. Therefore, we want to transform this sophisticated NFPP into a easily solvable problem.

Theorem 9 *The NFPP in Definition 27 is equivalent to a polynomial programming problem with polynomial constraints (PPPPC).*

$$\max_{\alpha,\beta,\omega_1,\ldots,\omega_N,\pi} \pi f(\alpha, \beta, \omega_1, \ldots, \omega_N) \tag{3.26}$$

subject to $\alpha{+}\beta = 1$, $\alpha, \beta \geqslant 0$, $\sum_{i=1}^{N} \omega_i = 1$, $\omega_i \geqslant 0$, $i = 1, \cdots, N$, $0 \leqslant \pi \leqslant 1/g(\alpha, \beta, \omega_1, \ldots, \omega_N)$.

Proof. If $(\overline{\alpha},\overline{\beta},\overline{\omega}_1,\ldots,\overline{\omega}_N,\overline{\pi})$ *is an optimal solution of PPPPC, then* $\overline{\pi}=1/g(\overline{\alpha},\overline{\beta},\overline{\omega}_1,\ldots,\overline{\omega}_N)$. *Thus* $\overline{\pi}f(\overline{\alpha},\overline{\beta},\overline{\omega}_1,\ldots,\overline{\omega}_N)=f(\overline{\alpha},\overline{\beta},\overline{\omega}_1,\ldots,\overline{\omega}_N)/g(\overline{\alpha},\overline{\beta},\overline{\omega}_1,\ldots,\overline{\omega}_N)$. *For any feasible solution* $(\alpha,\beta,\omega_1,\ldots,\omega_N)$ *of NFPP, the constraints of PPPPC are satisfied by setting* $\pi=1/g(\alpha,\beta,\omega_1,\ldots,\omega_N)$, *so* $\pi f(\alpha,\beta,\omega_1,\ldots,\omega_N)\leqslant\overline{\pi}f(\overline{\alpha},\overline{\beta},\overline{\omega}_1,\ldots,\overline{\omega}_N)$, *i.e.* $f(\alpha,\beta,\omega_1,\ldots,\omega_N)/g(\alpha,\beta,\omega_1,\ldots,\omega_N)\leqslant f(\overline{\alpha},\overline{\beta},\overline{\omega}_1,\ldots,\overline{\omega}_N)/g(\overline{\alpha},\overline{\beta},\overline{\omega}_1,\ldots,\overline{\omega}_N)$.

Conversely, if $(\overline{\alpha},\overline{\beta},\overline{\omega}_1,\ldots,\overline{\omega}_N)$ *solves NFPP, then for any feasible solution* $(\alpha,\beta,\omega_1,\ldots,\omega_N,\pi)$ *of PPPPC we have* $\pi f(\alpha,\beta,\omega_1,\ldots,\omega_N)\leqslant f(\alpha,\beta,\omega_1,\ldots,\omega_N)/g(\alpha,\beta,\omega_1,\ldots,\omega_N)\leqslant f(\overline{\alpha},\overline{\beta},\overline{\omega}_1,\ldots,\overline{\omega}_N)/g(\overline{\alpha},\overline{\beta},\overline{\omega}_1,\ldots,\overline{\omega}_N)=\overline{\pi}f(\overline{\alpha},\overline{\beta},\overline{\omega}_1,\ldots,\overline{\omega}_N)$ *with* $\overline{\pi}=1/g(\overline{\alpha},\overline{\beta},\overline{\omega}_1,\ldots,\overline{\omega}_N)$.

Although PPPPC is a polynomial programming problem, the polynomial constraints make it very hard to solve. We further simplify it as an nonlinear parametric programming problem (NPPP).

Definition 14 *[Nonlinear Parametric Programming Problem] Let* $f(\alpha,\beta,\omega_1,\ldots,\omega_N)=\sum_{i=1}^{m}a_i(\alpha)^{b_i}$
$(\beta)^{c_i}\prod_{j=1}^{N}(\omega_j)^{d_{ij}}$ *and* $g(\alpha,\beta,\omega_1,\ldots,\omega_N)=\sum_{i=1}^{n}o_i(\alpha)^{p_i}(\beta)^{q_i}\prod_{j=1}^{N}(\omega_j)^{r_{ij}}$, *the NPPP is defined below.*

$$F(\gamma)=\max_{\alpha,\beta,\omega_1,\ldots,\omega_N} f(\alpha,\beta,\omega_1,\ldots,\omega_N)-\gamma g(\alpha,\beta,\omega_1,\ldots,\omega_N) \tag{3.27}$$

subject to $\alpha+\beta=1$, $\alpha,\beta\geqslant 0$, $\sum_{i=1}^{N}\omega_i=1$, $\omega_i\geqslant 0$, $i=1,\cdots,N$.

Theorem 10 *The NFPP in Definition 27 is equivalent to the NPPP in Definition 18, i.e.,* $\gamma=\max_{\alpha,\beta,\omega_1,\ldots,\omega_N}\frac{f(\alpha,\beta,\omega_1,\ldots,\omega_N)}{g(\alpha,\beta,\omega_1,\ldots,\omega_N)}$ *iff* $F(\gamma)=\max_{\alpha,\beta,\omega_1,\ldots,\omega_N} f(\alpha,\beta,\omega_1,\ldots,\omega_N)-\gamma g(\alpha,\beta,\omega_1,\ldots,\omega_N)=0$.

Proof. If $(\overline{\alpha},\overline{\beta},\overline{\omega}_1,\ldots,\overline{\omega}_N)$ *is a feasible solution of* $F(\gamma)=0$, *then* $f(\overline{\alpha},\overline{\beta},\overline{\omega}_1,\ldots,\overline{\omega}_N)-\gamma g(\overline{\alpha},\overline{\beta},\overline{\omega}_1,\ldots,\overline{\omega}_N)=0$. *Thus* $f(\alpha,\omega_1,\ldots,\omega_N)-\gamma g(\alpha,\omega_1,\ldots,\omega_N)\leqslant f(\overline{\alpha},\overline{\beta},\overline{\omega}_1,\ldots,\overline{\omega}_N)-\gamma g(\overline{\alpha},\overline{\beta},\overline{\omega}_1,\ldots,\overline{\omega}_N)=0$. *We have* $\gamma=f(\overline{\alpha},\overline{\beta},\overline{\omega}_1,\ldots,\overline{\omega}_N)/g(\overline{\alpha},\overline{\beta},\overline{\omega}_1,\ldots,\overline{\omega}_N)\geqslant f(\alpha,\omega_1,\ldots,\omega_N)/g(\alpha,\omega_1,\ldots,\omega_N)$. *Therefore,* γ *is a maximum value of NFPP and* $(\overline{\alpha},\overline{\beta},\overline{\omega}_1,\ldots,\overline{\omega}_N)$ *is an optimal solution of NFPP.*

Conversely, if $(\overline{\alpha}, \overline{\beta}, \overline{\omega}_1, \ldots, \overline{\omega}_N)$ *solves NFPP, then we have* $\gamma = f(\overline{\alpha}, \overline{\beta}, \overline{\omega}_1, \ldots, \overline{\omega}_N)/g(\overline{\alpha}, \overline{\beta}, \overline{\omega}_1, \ldots, \overline{\omega}_N) \geqslant f(\alpha, \omega_1, \ldots, \omega_N)/g(\alpha, \omega_1, \ldots, \omega_N)$. *Thus* $f(\alpha, \omega_1, \ldots, \omega_N) - \gamma g(\alpha, \omega_1, \ldots, \omega_N) \leqslant f(\overline{\alpha}, \overline{\beta}, \overline{\omega}_1, \ldots, \overline{\omega}_N) - \gamma g(\overline{\alpha}, \overline{\beta}, \overline{\omega}_1, \ldots, \overline{\omega}_N) = 0$. *We have* $F(\gamma) = 0$ *and the maximum is taken at* $(\overline{\alpha}, \overline{\beta}, \overline{\omega}_1, \ldots, \overline{\omega}_N)$.

Now the original NFPP has been successfully transformed into the straightforward NPPP. This transformation can efficiently speed up the classification convergence due to the following properties.

Theorem 11 $F(\gamma)$ *is convex.*

Proof: Suppose that $(\overline{\alpha}, \overline{\beta}, \overline{\omega}_1, \ldots, \overline{\omega}_N)$ *is an optimal solution of* $F((1-\lambda)\gamma_1 + \lambda\gamma_2)$ *with* $\gamma_1 \neq \gamma_2$ *and* $0 \leqslant \lambda \leqslant 1$. $F((1-\lambda)\gamma_1 + \lambda\gamma_2) = f(\overline{\alpha}, \overline{\beta}, \overline{\omega}_1, \ldots, \overline{\omega}_N) - ((1-\lambda)\gamma_1 + \lambda\gamma_2) g(\overline{\alpha}, \overline{\beta}, \overline{\omega}_1, \ldots, \overline{\omega}_N) = \lambda(f(\overline{\alpha}, \overline{\beta}, \overline{\omega}_1, \ldots, \overline{\omega}_N) - \gamma_2 g(\overline{\alpha}, \overline{\beta}, \overline{\omega}_1, \ldots, \overline{\omega}_N)) + (1-\lambda)(f(\overline{\alpha}, \overline{\beta}, \overline{\omega}_1, \ldots, \overline{\omega}_N) - \gamma_1 g(\overline{\alpha}, \overline{\beta}, \overline{\omega}_1, \ldots, \overline{\omega}_N)) \leqslant \lambda \max_{\alpha, \beta, \omega_1, \ldots, \omega_N} f(\alpha, \beta, \omega_1, \ldots, \omega_N) - \gamma_2 g(\alpha, \beta, \omega_1, \ldots, \omega_N) + (1-\lambda) \max_{\alpha, \beta, \omega_1, \ldots, \omega_N} f(\alpha, \beta, \omega_1, \ldots, \omega_N) - \gamma_1 g(\alpha, \beta, \omega_1, \ldots, \omega_N) = \lambda F(\gamma_2) + (1-\lambda) F(\gamma_1)$. *Thus,* $F(\gamma)$ *is convex.*

Theorem 12 $F(\gamma)$ *is monotonically decreasing.*

Proof: Suppose that $\gamma_1 > \gamma_2$ *and* $(\overline{\alpha}, \overline{\beta}, \overline{\omega}_1, \ldots, \overline{\omega}_N)$ *is an optimal solution of* $F(\gamma_1)$. *Thus,* $F(\gamma_1) = f(\overline{\alpha}, \overline{\beta}, \overline{\omega}_1, \ldots, \overline{\omega}_N) - \gamma_1 g(\overline{\alpha}, \overline{\beta}, \overline{\omega}_1, \ldots, \overline{\omega}_N) < f(\overline{\alpha}, \overline{\beta}, \overline{\omega}_1, \ldots, \overline{\omega}_N) - \gamma_2 g(\overline{\alpha}, \overline{\beta}, \overline{\omega}_1, \ldots, \overline{\omega}_N) \leqslant \max_{\alpha, \beta, \omega_1, \ldots, \omega_N} f(\alpha, \beta, \omega_1, \ldots, \omega_N) - \gamma_2 g(\alpha, \beta, \omega_1, \ldots, \omega_N) = F(\gamma_2)$.

Theorem 13 $F(\gamma) = 0$ *has a unique solution.*

Proof: Based on the above-mentioned theorems, we know $F(\gamma)$ *is continuous as well as decreasing. In addition,* $\lim_{\gamma \to +\infty} F(\gamma) = -\infty$ *and* $\lim_{\gamma \to -\infty} F(\gamma) = +\infty$.

The procedure of solving this NPPP includes two parts: (1) find such a reasonable parameter γ ($F(\gamma) = 0$), making NPPP equivalent to NFPP; (2) given the parameter γ, solve a polynomial programming problem about the original variables $\alpha, \beta, \omega_1, \ldots, \omega_N$. Our weight adjustment mechanism is an iterative procedure to find the solution of $F(\gamma) = 0$

and the corresponding weights after each iteration of the classification process. We first generate an initial unified classification kernel $\mathbf{K}_j^{(1)}$ with equal weights of $\frac{1}{N}$ to produce an initial classification result on the collaboration multigraph. According to the initial classification result, we then calculate an initial $F(\gamma)$. Since $F(\gamma)$ is a monotonic decreasing function and $F(0) = \max_{\alpha,\beta,\omega_1,\ldots,\omega_N} f(\alpha,\beta,\omega_1,\ldots,\omega_N)$ is obviously non-negative, we start with an initial $\gamma = 0$ and solve the subproblem $F(0)$ by using existing fast polynomial programming model to update the weights $\alpha,\beta,\omega_1,\ldots,\omega_N$. The parameter γ is gradually increased by $\gamma = f(\alpha,\beta,\omega_1,\ldots,\omega_N)/g(\alpha,\beta,\omega_1,\ldots,\omega_N)$ to help the algorithm enter the next round. The algorithm repeats the above-mentioned iterative procedure until $F(\gamma)$ converges to 0.

By assembling different pieces together, we provide the pseudo code of our **AEClass** classifier in Algorithm 3.

3.5 Experimental Evaluation

We have performed extensive experiments to evaluate the performance of our **AEClass** classifier on real graph datasets.

3.5.1 Experimental Datasets

The first real-world dataset is extracted from the DBLP Bibliography data [1]. We build a coauthor graph with highly prolific 100, 000 authors from all research areas and 712, 834 associated links where vertices represent authors and edges represent their coauthor relationships, and two associated activity graphs: conference graph and term graph. According to [52], we categorize research areas into 24 fields: AI, AIGO, ARC, BIO, CV, DB, DIST, DM, EDU, GRP, HCI, IR, ML, MUL, NLP, NW, OS, PL, RT, SC, SE, SEC, SIM, WWW. We utilize a multi-typed soft clustering framework, NetClus [50], to cluster conferences and terms into 24 categories simultaneously. According to conference's or term's clustering distribution over 24 categories and ranking score in each category, we calculate the

[1] http://dblp.uni-trier.de/xml/

Algorithm 2 Activity-Edge Centric Multi-label Classification

Input: a collaboration graph CG, N activity graphs AG_i, a class number K, initial weights $\alpha^{(2)}=\beta^{(2)}=\frac{1}{2}$, $\omega_1^{(1)}=\cdots=\omega_N^{(1)}=\frac{1}{N}$ and a parameter $\gamma^{(1)}=0$.
Output: the predicted label Y_u for the testing instances V_l.

Invoke NetClus to partition each of N kinds of activities into K clusters simultaneously;
Calculate the category similarity on each AG_i in Eq.(3.11);
Execute the edge classification on CG based on each AG_i in Eqs.(3.1)-(3.2);
Construct the collaboration multigraph MG;
Compute the unified weight $\mathbf{W}_j^{(1)}$ of MG for each c_j in Eq.(3.3);
Calculate the transition probability $\mathbf{T}_j^{(1)}$ for each c_j in Eqs.(4.2)-(4.3);
Generate the classification kernel $\mathbf{K}_j^{(1)}$ for each c_j in Eq.(3.6);
for t=1 **to** $F(\gamma^{(t)})$ converges to 0
  Calculate the class-membership matrices $\mathbf{X}^{(t)}$ in Eqs.(3.8)-(3.10) and
  $\mathbf{Y}^{(t)}$ in Eqs.(3.12)-(3.13);
  Compute the Macro-F1 score in Eq.(3.22);
  Solve $F(\gamma^{(t)})$ in Eq.(4.22);
  Update $\omega_1^{(t+1)}$, ..., $\omega_N^{(t+1)}$ if t=1, or update $\alpha^{(t+1)}$, $\beta^{(t+1)}$, $\omega_1^{(t+1)}$, ...,
  $\omega_N^{(t+1)}$ if t¿1;
  Refine $\gamma^{(t+1)}=f(\alpha^{(t+1)},\beta^{(t+1)},\omega_1^{(t+1)},...,\omega_N^{(t+1)})/g(\alpha^{(t+1)},\beta^{(t+1)},\omega_1^{(t+1)},$
  $...,\omega_N^{(t+1)})$;
  Update $\mathbf{W}_j^{(t+1)}$ in Eq.(3.3);
  Adjust $\mathbf{T}_j^{(t+1)}$ in Eqs.(4.2)-(4.3);
  Update $\mathbf{K}_j^{(t+1)}$ in Eq.(3.17);
Return $\mathbf{Y}^{(t)}$ and Y_u.

similarities between conferences or terms. The classification goal is infer research areas of each author.

Last.fm [2] is a music-oriented online social network. We use the API call *user.getfriends* to collect the list of friends and construct a friendship graph with 50, 000 users and 496, 611 associated links where vertices represent users and edges denote their friendships. The t-wo activity networks: artist graph and track graph are generated by invoking the API calls *artist.getSimilar* and *track.getSimilar* respectively. By calling the API calls *user.getTopArtists* and *user.getTopTracks*, we classify each friendship edge in terms of the same artists or the same tracks shared by two users. The classification task is to assign each user to a subset

[2]http://www.last.fm/api

of 21 music genres in the database: acoustic, ambient, blues, classical, country, electronic, emo, folk, hardcore, hip hop, indie, jazz, latin, metal, pop, pop punk, punk, reggae, rnb, rock, soul.

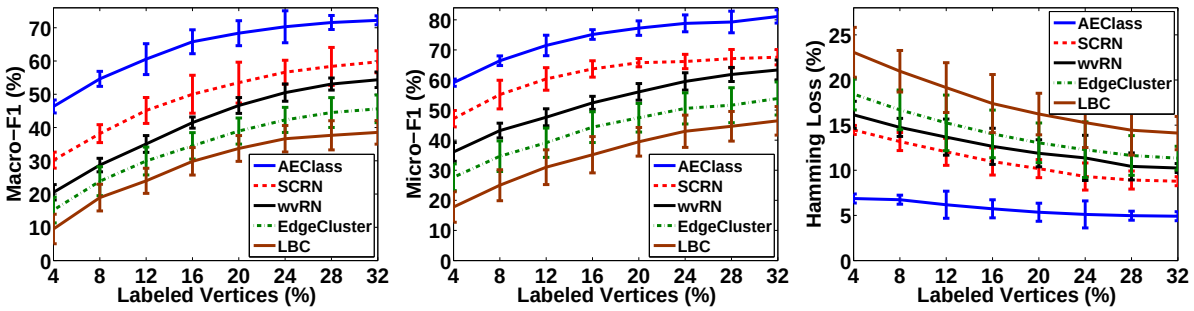
The third real dataset is extracted from the Internet Movie Database (IMDb) [3]. We construct a collaboration graph with 10,000 highly prolific actors and 270,227 links where vertices represent actors and edges specify their costar relationships in terms of co-appearance of actors in the same movies. We build a movie activity graph where edges denote co-direct relationship between movies, i.e., movies are directed by the same directors. The objective is to associate each actor with a subset of 22 movie genres: Action, Adventure, Animation, Biography, Comedy, Crime, Documentary, Drama, Family, Fantasy, Film-Noir, History, Horror, Music, Musical, Mystery, Romance, Sci-Fi, Sport, Thriller, War, Western.

3.5.2 Comparison Methods and Evaluation

We compare **AEClass** with two representative link-based classification algorithms, **LBC** [42], **wvRN** [43], and two recently developed multi-label classifiers, **EdgeCluster** [37], **SCRN** [40]. All four methods perform multi-label classification on a single weighted graph based on the assumption of vertex homophily. The detailed introductions for four methods are presented in Section 8.2. Note that LBC is originally a multi-class classifier. In order to compare all algorithms, we modify the last step in LBC and use the posterior probability distribution over K classes as the multi-label classification result. AEClass integrates multiple information networks into a unified multigraph with combining both the vertex-centric multi-label classification and the edge-centric multi-label classification based on vertex-edge homophily. It also integrates both the structure affinity and the label vicinity into a unified classifier through dynamic weight tuning mechanism.

Evaluation Measures We use three measures to evaluate the quality of classification results generated by different methods. The first measure is Macro-F1 defined in Defini-

[3] http://www.imdb.com/interfaces



(a) Macro-F1 (b) Micro-F1 (c) Hamming Loss

Figure 3.7: Classification Quality on DBLP

tion 11. Given the same definitions in Eq.(3.18), other two metrics are defined as follows.

$$Micro{-}F1 = \frac{2\sum_{j=1}^{K}\sum_{i=l+1}^{N_{CG}} \mathcal{I}(\hat{y}_i^j = 1) y_i^j}{\sum_{j=1}^{K}\sum_{i=l+1}^{N_{CG}} \mathcal{I}(\hat{y}_i^j = 1) + \sum_{j=1}^{K}\sum_{i=l+1}^{N_{CG}} y_i^j} \tag{3.28}$$

Micro-F1 [53] represents the harmonic mean of micro average of precision and recall. The larger the value, the better the quality.

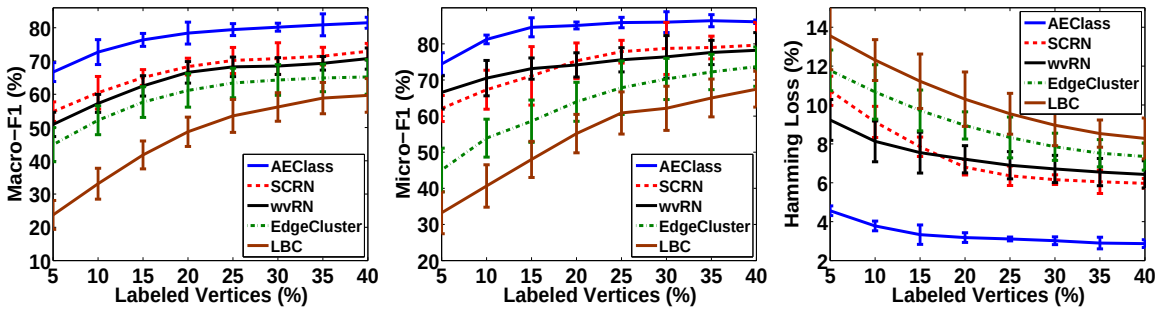
$$HammingLoss = \frac{1}{N_{CG} - l}\sum_{i=l+1}^{N_{CG}} \frac{1}{K} \|\mathcal{I}(\hat{y}_i = 1) \oplus y_i\|_1 \tag{3.29}$$

where $\oplus$ represents the XOR operation, and $\|\cdot\|_1$ specifies the $l1$-norm. Hamming Loss [54] measures the loss between true labels and predicted labels. The smaller the value, the better the quality.

3.5.3 Classification Quality

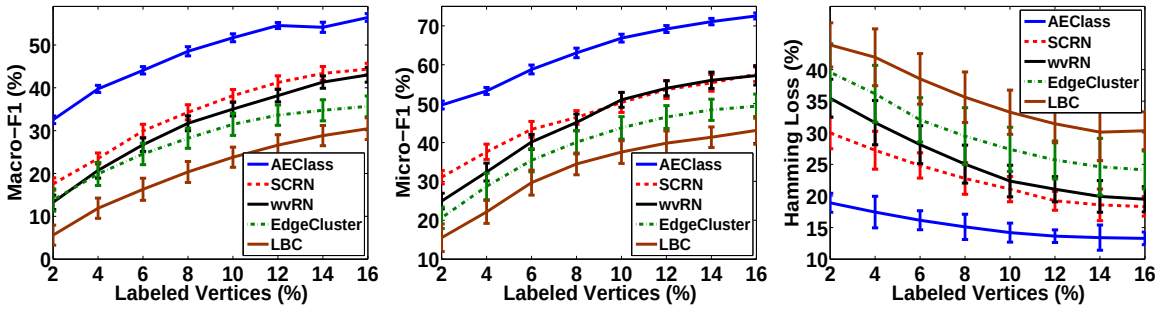
Figures 3.7-3.9 exhibit the classification quality on DBLP, Last.fm and IMDb by varying the proportion of labeled vertices respectively. For each proportion of labeled vertices, we average the performance scores over 10 cross-validation folds. The average performance scores with standard deviations of five multi-label classification methods are reported with respect to three evaluation measures of Macro-F1, Micro-F1 and Hamming Loss. We make the following observations on the performances by different methods.

First, AEClass, SCRN and wvRN significantly outperform LBC and EdgeCluster on all



(a) Macro-F1 (b) Micro-F1 (c) Hamming Loss

Figure 3.8: Classification Quality on Last.fm



(a) Macro-F1 (b) Micro-F1 (c) Hamming Loss

Figure 3.9: Classification Quality on IMDb

three evaluation measures. We first categorize five multi-label classification methods into non-transductive learning methods and transductive learning methods, based on how they utilize topological structure information. As non-transductive learning methods, both LBC and EdgeCluster only utilize the direct links between vertices in the graph, i.e., one-hop structure information, to produce vertex's features. As transductive learning approaches, AEClass, SCRN and wvRN make full use of both direct links and indirect edges (the circle of friends) between vertices through iterative graph propagation, i.e., multiple-hop structure information, to further improve the classification quality. These results demonstrate the importance of exploiting both direct links and indirect edges for multi-label classification in networked data.

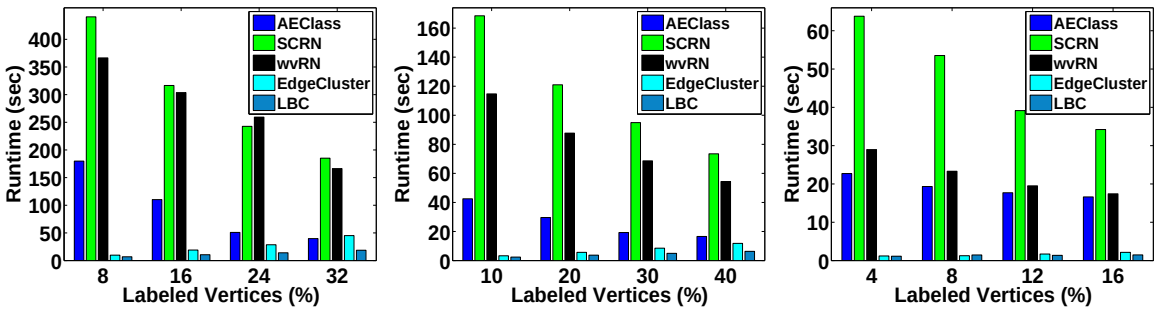
Second, SCRN always outperforms wvRN on three graph datasets. Although SCRN and wvRN exploit the very similar relational inference framework, SCRN improves wvRN

by integrating both the network topology and the social context features extracted by EdgeCluster into the classifier. A careful examination reveals that these two approaches are very close in terms of prediction performance in many situations, in spite of the optimization adopted by SCRN. A reasonable explanation is that both of them are only based on the assumption of vertex homophily, i.e., the principle that similar vertices in nature are connected to each other with social links.

Finally, among all five classification methods, AEClass achieves the best classification performance on all three real datasets for all three evaluation measures. Compared to other algorithms, AEClass averagely achieves 14.6% Macro-F1 increase, 12.1% Micro-F1 boost and 5.2% loss reduction on DBLP, 10.2% Macro-F1 growth, 9.9% Micro-F1 increase and 4.1% loss decrease on Last.fm, and 16.7% Macro-F1 increase, 16.2% Micro-F1 boost and 7.5% loss reduction on IMDb, respectively. Note that even if the proportion of labeled vertices is very small, such as 2% and 4%, AEClass still can achieve comparable accuracy on all datasets. Concretely, there are four critical reasons for high accuracy of AEClass: (1) the structure information from associated activity networks boosts the effectiveness of classification. Activity network partition provides us with additional activity labels; (2) the multigraph organization integrates both the vertex-centric multi-label classification based on vertex homophily and the edge-centric multi-label classification based on vertex-edge homophily to leverage the classification performance; (3) Activity network partition captures the inter-dependencies among multiple class labels; and (4) the iterative learning algorithm help the classifier achieve a good balance among different activity-based edge classification schemes and an effective integration of the structure affinity and the label vicinity.

3.5.4 Classification Efficiency

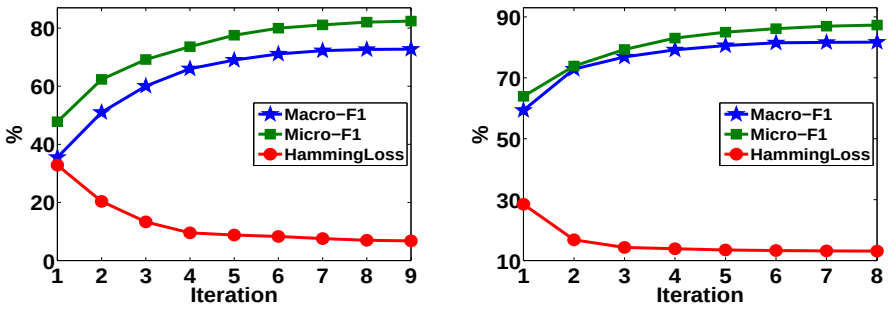
Figures 6.5 (a), (b) and (c) present the classification time on DBLP, Last.fm and IMDb with different proportions of labeled vertices respectively. First, LBC has lowest runtime



(a) DBLP (b) Last.fm (c) IMDb

Figure 3.10: Classification Efficiency

in seconds compared to all other algorithms in all experiments, since it is a logistic regression classifier by aggregating the labels of neighbors as vertex's feature vector. Second, EdgeCluster, a linear SVM classifier with an edge clustering scheme to extract sparse social dimensions, is slightly slower than LBC since the linear SVM approaches generally fall behind the LR methods in speed. Both LBC and EdgeCluster are faster than other three methods because both only utilize the direct links between vertices, i.e., one-hop structure information. In comparison, AEClass, SCRN and wvRN use both direct links and indirect edges (the circle of friends) between vertices, i.e., multiple-hop structure information. Thus, the last three classifiers have higher time complexity than the first two models but they achieve better classification quality. Third, wvRN is consistently faster than SCRN on all three datasets. SCRN improves wvRN by integrating the social dimensions extracted by EdgeCluster into the classifier. This improvement results in an additional computational cost for calculating the class propagation probability of each vertex on each class. Finally, AEClass significantly outperforms the other two transductive learning based multi-label classifiers: SCRN and wvRN. Although SCRN and wvRN execute the classification on a general graph, AEClass does classification on an activity-based collaboration multigraph by augmenting its edges with class labels from each activity graph. There are three main reasons for high efficiency of AEClass: (1) the multigraph organization increases the size of dataset but reduces the computational cost of classification. As we discussed in Subsection 3.4.2, based on the vertex homophily, no matter which class the current objective is,



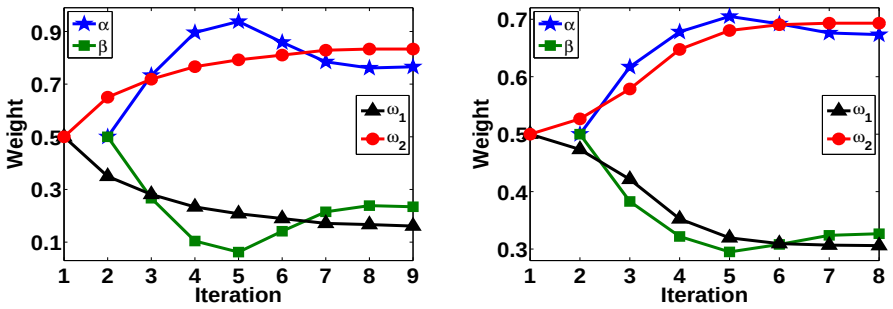
(a) DBLP (b) Last.fm

Figure 3.11: Classification Convergence

both SCRN and wvRN need to check each neighbor of a vertex and summarize the labels of all neighbors. In contrast, AEClass only picks up those vertices and links with the same label as the current objective class to execute the inference. Most importantly, AEClass stops the label propagation through irrelevant neighbors (without a link with the same label as the current objective class) in the future iterations; (2) the label vicinity between vertices based on the class-membership distribution over K classes is integrated into the classifier; and (3) we transform the original nonlinear fractional programming problem of multiple weights into a nonlinear parametric programming problem of single variable. According to Theorems 14-17, solving $F(\gamma)$ for a given γ is a polynomial programming problem which can be sped up by existing fast polynomial programming model.

3.5.5 Classification Convergence

Figure 4.13 (a) and (b) exhibit the trend of classification convergence in terms of Macro-F1, Micro-F1, and Hamming Loss on DBLP with 4% label nodes and Last.fm with 5% label vertices. Both the Macro-F1 values and the Micro-F1 scores in two figures keep increasing and have concave curves when we iteratively perform the tasks of vertex labeling, weight update and kernel adjustment during the classification process. On the other hand, the Hamming Loss values decrease with the classification iterations and have a convex curve. The classification process converges very quickly, usually in eight iterations for Last.fm and nine iterations for DBLP.



(a) DBLP (b) Last.fm

Figure 3.12: Weight Update

Figure 3.12 (a) and (b) show the tendency of weight update on DBLP and Last.fm respectively. α and β in two figures represent the weights of structure affinity and label vicinity in the unified classifier $\mathbf{K}_j^{(t+1)}$ in Eq.(3.17) respectively. ω_1 and ω_2 in Figure 3.12 (a) denote the weights of the conference graph and the term graph respectively. ω_1 and ω_2 in Figure 3.12 (b) represent the weights of the artist graph and the track graph respectively. We keep the constraints α+β=1 and ω_1+ω_2=1 during the classification process. We observe that all the weights converge as the clustering process converges. An interesting phenomenon is that α first increases and then decreases with the iterations and the β curve has a converse trend. A reasonable explanation is that there is lack of enough labeling information at the beginning of classification such that the unified classifier has to rely mostly on the structure affinity to achieve a good classification performance. After a few iterations, we have enough labeling information to utilize both the structure affinity and the label vicinity to classify vertices. An interesting finding is that the term weight is increasing but the conference weight is decreasing with more iterations. A reasonable explanation is that people who have many publications on the same conferences may have different research topics but people who have many papers with the same terms usually have the same research topics. For example, both database papers and data mining papers are published on *VLDB*. Similarly, the track weight increases but the artist weight decreases with more iterations. This is because users who favor the same artists may belong to different music genres since the artists are often related to multiple genres but users who like the same

Table 3.1: Class-membership Probabilities of Authors Based on Conference and Keyword Partitions from DBLP

Author/Class	DB	DM	ML	IR
Peter L. Bartlett	0.019	0.039	0.938	0.004
Elisa Bertino	0.738	0.054	0.028	0.180
Andrei Z. Broder	0.037	0.097	0.015	0.851
Michael J. Carey	0.965	0.029	0.006	0.023
W. Bruce Croft	0.054	0.007	0.037	0.902
David J. DeWitt	0.912	0.057	0.004	0.027
Inderjit S. Dhillon	0.030	0.409	0.457	0.104
Christos Faloutsos	0.321	0.500	0.031	0.147
Jiawei Han	0.391	0.463	0.045	0.100
H. V. Jagadish	0.850	0.056	0.009	0.048
Michael I. Jordan	0.007	0.062	0.917	0.014
Daphne Koller	0.026	0.045	0.915	0.013
Vipin Kumar	0.120	0.622	0.199	0.059
Bing Liu	0.086	0.427	0.266	0.220
Hector Garcia-Molina	0.788	0.010	0.016	0.186
C. J. van Rijsbergen	0.003	0.051	0.024	0.922
Michael Stonebraker	0.946	0.013	0.007	0.034
Jeffrey D. Ullman	0.824	0.065	0.064	0.047
Philip S. Yu	0.342	0.496	0.044	0.118
Mohammed J. Zaki	0.148	0.672	0.057	0.123

tracks usually belong to the same music genres.

3.5.6 Case Study

We examine some details of the experiment results on DBLP 100,000 Authors when the proportion of labeled vertices is equal to 32% based on the coauthor graph, the conference graph and the term graph. Table 3.1 shows the set of authors and their class-membership probabilities after seven iterations based on 24 conference categories and 24 term categories. We only present most prolific DBLP experts in the area of database (DB), data mining (DM), machine learning (ML) and information retrieval (IR). The class-membership scores in Table 3.1 are normalized by different (conference or term) categories for each author. We observe that the predicted class memberships of authors are consistent with their actual research areas. For those experts with unique research areas, such as Michael J. Carey and Michael Stonebraker, the primary research areas for them in the predicted result

are obviously consistent with their actual research areas; For those researchers known to work in multiple research areas, the predicted class-membership distributions also correspond to their current research activities. For example, both Jiawei Han and Philip S. Yu are experts on data mining and database, though their DM probabilities are slightly higher since each of them and their circle of co-authors have more DM papers. This table also shows that each author has a class-membership score in each category. This demonstrates that our AEClass model can make each author quickly reach each class label.

3.6 Conclusions

We have presented an edge-centric multi-label classification approach for mining heterogeneous information networks. First, we integrate the primary social network and multiple associated activity networks into a unified multigraph with edge classification. Second, we combine both the structure affinity and the label vicinity based on multiple activity networks into a unified classifier. Third, an iterative learning algorithm is proposed dynamically refine the classification result by continuously adjusting the weights on different activity-based edge classification schemes from multiple activity graphs, while constantly learning the contributions of the structure affinity and the label vicinity in the unified classifier.

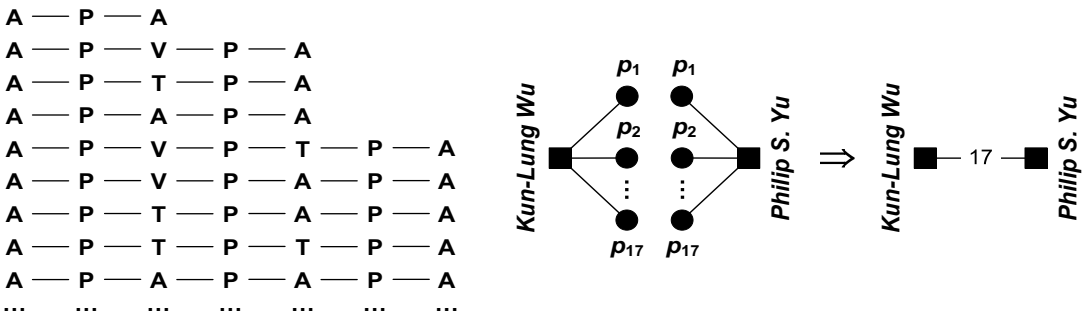
CHAPTER 4

VEPATHCLUSTER: INTEGRATING VERTEX-CENTRIC CLUSTERING WITH EDGE-CENTRIC CLUSTERING FOR META PATH GRAPH ANALYSIS

Meta paths are good mechanisms to improve the quality of graph analysis on heterogeneous information networks. This chapter presents a meta path graph clustering framework, VEPathCluster, that combines meta path vertex-centric clustering with meta path edge-centric clustering for improving the clustering quality of heterogeneous networks. First, we propose an edge-centric path graph model to capture the meta-path dependencies between pairwise path edges. We model a heterogeneous network containing M types of meta paths as M vertex-centric path graphs and M edge-centric path graphs. Second, we propose a clustering-based multigraph model to capture the fine-grained clustering-based relationships between pairwise vertices and between pairwise path edges. We perform clustering analysis on both a unified vertex-centric path graph and each edge-centric path graph to generate vertex clustering and edge clusterings of the original heterogeneous network respectively. Third, a reinforcement algorithm is provided to tightly integrate vertex-centric clustering and edge-centric clustering by mutually enhancing each other. Finally, an iterative learning strategy is presented to dynamically refine both vertex-centric clustering and edge-centric clustering by continuously learning the contributions and adjusting the weights of different path graphs.

4.1 Introduction

Heterogeneous information networks are graphs with heterogeneous types of entities and links. A meta path is a path connecting multiple types of entities through a sequence of heterogeneous meta links, representing different kinds of semantic relations among different types of entities. DBLP dataset has four types of entities: authors (A), publishing venues



(a) Meta Paths between Authors (b) Example Vertex-centric Path Graph

Figure 4.1: Example Meta Paths and Path Graphs from DBLP

(V), papers (P) and paper terms (T). Figure 5.3 (a) gives nine example meta paths between authors in the DBLP dataset, each is composed of three types of meta links: A-P, V-P and T-P, representing different types of relationships between authors. More meta paths between authors can be generated through link combination and propagation. The meta path A-P-A captures the coauthor relationship, whereas the path A-P-V-P-A represents the relationship between a pair of authors through their papers published on the common venues. For each type of meta paths, we can construct a vertex-centric path graph to capture an individual type of relationships between authors. For example, Figure 5.3 (b) shows that we join one type of links (A-P) and its opposite form (P-A) to generate a vertex-centric A-P-A path graph, where vertices represent authors and edges denote the coauthor relationships between authors. For each pair of coauthors, say *Kun-Lung Wu* and *Philip S. Yu*, we can represent the A-P-A path by using parallel edges, each representing one of their coauthored papers ($p_1, \cdots, p_{17}$). By join composition, we obtain the total number of their coauthored papers (17). Clearly, mining heterogeneous information networks through multiple path graphs can provide new insights about how ideas and opinions on different subjects propagate differently among the same set of people.

Meta path-based social network analysis is gaining attention in recent years [55, 56, 49, 22, 57, 58]. Existing efforts utilize a selection of meta paths between the same type of entities to improve the quality of similarity search, classification, clustering, link prediction and citation recommendation in heterogeneous networks. However, none of the existing

methods have addressed all of the following challenges.

- **Vertex-centric clustering w.r.t. multiple path graphs.** As shown in Figure 5.3, different meta paths exhibit different semantic meanings about the same type of entities. Thus, the vertex clustering results based on different path graphs are typically not identical. It is critical to develop a unified clustering model that can efficiently integrate the clustering results from multiple path graphs and improve the overall clustering quality. Specifically, a dynamic weight assignment scheme should be employed to assign different weights to different path graphs to reflect their possibly different contributions towards the clustering convergence.

- **Fine-grained vertex assignment and clustering objective.** Meta-path graph analysis differentiates the semantics carried by different meta paths in a heterogeneous network. Consequently, it demands fine-grained vertex assignment and clustering objective to further improve the clustering quality. However, existing partitioning clustering approaches, such as K-Means and K-Medoids [59], usually assign each vertex to its closest center. We argue that this kind of vertex assignment may not always produce an accurate clustering result. Consider Figure 4.2 (a), by performing K-Means on the A-P-A path graph to assign *Kun-Lung Wu* to two centers of *Bugra Gedik* and *Philip S. Yu*, Figure 4.2 (b) shows a vertex assignment, i.e., by simply using the coarse path edge weight (the total number of coauthored papers) to measure vertex closeness, *Kun-Lung Wu* and *Philip S. Yu* are closer than *Kun-Lung Wu* and *Bugra Gedik*. However, in reality, *Kun-Lung Wu* and *Bugra Gedik* are known as database researchers with no or very few data mining papers but *Philip S. Yu* is a well-known expert on data mining with much more data mining papers than database publications, thus the vertex assignment in Figure 4.2 (c) is more accurate and better quality. This is because the similarity measures used in vertex assignment and clustering objective of existing methods are too coarse to reflect the above ground truth.

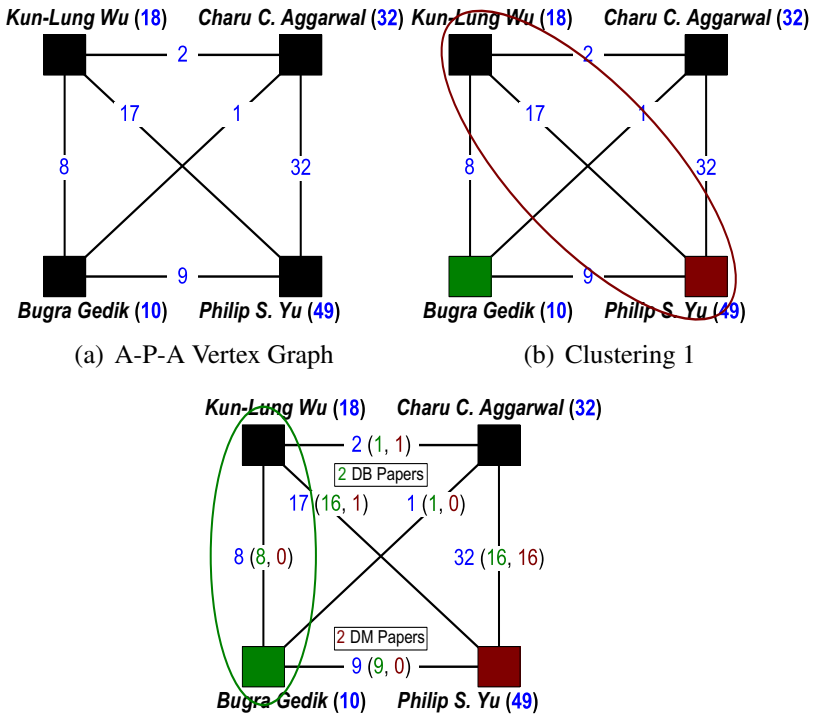
- **Edge-centric clustering w.r.t. multiple path graphs.** Conventional graph clustering models are usually based on the existence of vertex homophily. However, we argue that vertex homophily without edge clustering is insufficient for meta-path graph analysis on heterogeneous networks. Consider Figures 4.2 (b) and (c) again, there is only one of 17 coauthored papers between *Kun-Lung Wu* and *Philip S. Yu* published on DM conference (*KDD*) but all 8 coauthored papers between *Kun-Lung Wu* and *Bugra Gedik* are published on DB conferences, indicating that *Kun-Lung Wu*, *Bugra Gedik* and the path edge between them belong to cluster *DB* with very high probability. In comparison, it is highly probable that *Philip S. Yu* and the path edge between *Kun-Lung Wu* and *Philip S. Yu* belong to different clusters. Without considering edge clustering, the vertex homophily alone can lead to inaccurate vertex clustering.

- **Integrating vertex-centric clustering and edge-centric clustering.** Vertex clustering and edge clustering on heterogeneous networks may have individual clustering goals and due to the different semantic relationships implied by different meta paths. Relying on either of them alone may result in incomplete and possibly inaccurate clustering results. However, none of existing methods study how to effectively combine the above two techniques into a unified meta path graph clustering model.

To address the above challenges, we develop an efficient vertex/edge-centric meta path graph clustering approach, **VEPathCluster**, with four original contributions.

- We model a heterogeneous network containing multiple types of meta paths in terms of multiple vertex-centric path graphs and multiple edge-centric path graphs. Each meta path corresponds to one vertex-centric path graph and one edge-centric path graph.

- We propose a clustering-based multigraph model to capture the fine-grained clustering-based relationships between pairwise vertices and between pairwise path edges about given K clusters.



(a) A-P-A Vertex Graph (b) Clustering 1

(c) Clustering 2

Figure 4.2: Coarse Vertex Assignment/Clustering Objective

- We integrate multiple types of vertex-centric path graphs with different semantics into a unified vertex-centric path graph in terms of their contributions towards the clustering objective. We cluster both the unified vertex-centric path graph and each edge-centric path graph to generate vertex clustering and edge clusterings of the original heterogeneous network respectively.

- We design a reinforcement algorithm to tightly integrate vertex-centric clustering and edge-centric clustering by mutually enhancing each other: (1) good vertex-centric clustering promotes good edge-centric clustering and (2) good edge-centric clustering elevates good vertex-centric clustering. We devise an iterative learning method to dynamically refine both vertex-centric clustering and edge-centric clustering by continuously learning the contributions and adjusting the weights of different path graphs.

- Empirical evaluation over real datasets demonstrates the competitiveness of VEPath-

CLUSTER against the state-of-the-art methods.

4.2 Related Work

Meta path-based social network analysis is gaining attention in recent years [55, 56, 49, 22, 57, 58]. PathSim [55] presented a meta path-based similarity measure for heterogeneous graphs. [56] proposed a meta path-based ranking model to find entities with high similarity to a given query entity. HCC [49] is a meta-path based heterogeneous collective classification method. PathSelClus [22] utilizes user guidance as seeds in some of the clusters to automatically learn the best weights for each meta-path in the clustering. MLI [57] is a multi-network link prediction framework by extracting useful features from multiple meta paths.

Graph clustering has been extensively studied in recent years [19, 13, 14, 50, 20, 15, 28, 60, 61, 62, 21, 23, 63, 51, 64, 65, 66]. Shiga et al. [19] presented a clustering method which integrates numerical vectors with modularity into a spectral relaxation problem. S-CAN [13] is a structural clustering algorithm to detect clusters, hubs and outliers in networks. MLR-MCL [14] is a multi-level graph clustering algorithm using flows to deliver significant improvements in both quality and speed. TopGC [15] is a fast algorithm to probabilistically search large, edge weighted, directed graphs for their best clusters in linear time. BAGC [21] constructs a Bayesian probabilistic model to capture both structural and attribute aspects of graph. GenClus [23] proposed a model-based method for clustering heterogeneous networks with different link types and different attribute types. CGC [63] is a multi-domain graph clustering model to utilize cross-domain relationship as co-regularizing penalty to guide the search of consensus clustering structure. FocusCO [64] solves the problem of finding focused clusters and outliers in large attributed graphs.

To the best of our knowledge, VEPATHCLUSTER is the first one to tightly integrate vertex-centric clustering and edge-centric clustering by mutually enhancing each other with combining different types of meta paths over heterogeneous information network.

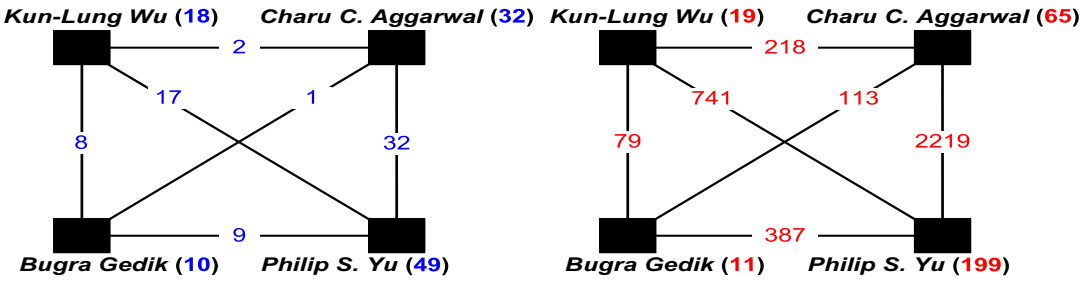
4.3 Problem Definition

We define the problem of vertex/edge-centric meta path graph clustering in terms of the following four concepts.

A heterogeneous information network is denoted as $G = (V, E)$, where V is the set of heterogeneous entity vertices in G, consisting of s types of entity vertices, i.e., $V = \bigcup_{i=1}^{s} V_i$, each V_i $(1 \leq i \leq s)$ represents the i^{th} types of entity vertices. E is the set of heterogeneous meta links denoting the relationships between entity vertices in V. Due to heterogeneous entity vertices with s types, E can be divided into $s \times s$ subsets E_{ij} $(1 \leq i, j \leq s)$ such that $E = \bigcup_{i=1,j=1}^{s} E_{ij}$, where E_{ij} is the set of meta links connecting vertices of the i^{th} type (V_i) to vertices of the j^{th} type (V_j). E_{ji} is the opposite form of E_{ij}, specifying the set of meta links from V_j to V_i.

The m^{th} meta path of length l, denoted by $MP_m =< E_{a_0a_1}, E_{a_1a_2}, \cdots, E_{a_{l-1}a_l} >$, is a sequence of different types of meta links, with source vertex type V_{a_0} and destination vertex type V_{a_l} $(1 \leq a_0, a_1, \cdots, a_l \leq s)$, such that $< E_{a_0a_1}, E_{a_1a_2}, \cdots, E_{a_{l-1}a_l} >$ are l meta link types connected through join composition. For example, meta path A-P-A is of length 2 and comprises two meta link types: A-P and P-A.

For each meta path in G, we construct a vertex-centric path graph to capture the meta-path based relationships between vertices. Formally, a vertex-centric path graph for MP_m is denoted as $VG_m = (V_{a_0}, V_{a_l}, E_m)$, where $V_{a_0} \in V$ is the set of source vertices and $V_{a_l} \in V$ is the set of destination vertices in MP_m, and $E_m \in E$ is the set of path edges between V_{a_0} and V_{a_l}. For the path edge set E_m, we compute its adjacency matrix $\mathbf{P}_m$ by multiplying adjacency matrix of each type of composite meta links $E_{a_0a_1}, E_{a_1a_2}, \cdots, E_{a_{l-1}a_l}$, denoted by $\mathbf{W}_{a_0a_1}, \mathbf{W}_{a_1a_2}, \cdots, \mathbf{W}_{a_{l-1}a_l}$ respectively. For Figure 5.3 (b), we use $\mathbf{W}_{AP}$ and $\mathbf{W}_{PA}$ to denote the adjacency matrices of two types of meta links A-P and P-A respectively. We calculate an adjacency matrix $\mathbf{P}_{AA} = \mathbf{W}_{AP} \times \mathbf{W}_{PA}$ to obtain the path edge between *Kun-Lung Wu* and *Philip S. Yu* with a value of 17. For presentation brevity, when the type of source



(a) A-P-A Path Graph (b) A-P-V-P-A Path Graph

Figure 4.3: Vertex-centric Path Graph

vertices is the same as the type of destination vertices in VG_m, i.e., $V_{a_0} = V_{a_l} = V_c \in V$, we simplify $VG_m = (V_{a_0}, V_{a_l}, E_m)$ as $VG_m = (V_c, E_m)$, and path edges in E_m measure the pairwise closeness between vertices in V_c. We denote the size of V_c as $N_{V_c} = |V_c|$ and denote the size of E_m as $N_{E_m} = |E_m|$.

In VEPathCluster, for a specific clustering task, users can select a subset of entity vertices of a certain type as the set of target vertices, denoted by V_c, and a subset of M target meta paths MP_m. We construct M vertex-centric path graphs VG_m. The problem of **Vertex/Edge-centric meta Path graph Clustering** (VEPathCluster) is to simultaneously perform two clustering tasks: (1) assign all entity vertices in V_c to K soft clusters with an $N_{V_c} \times K$ clustering membership matrix $\mathbf{X}$ with each row summing to 1, and (2) cluster all path edges in each E_m ($1 \leq m \leq M$) into K soft clusters with an $N_{E_m} \times K$ clustering membership matrix $\mathbf{Y}_m$ with each row summing to 1. The desired clustering result should achieve the two goals: (1) both path edges and their associated vertices should belong to the same clusters, and vertices within each cluster are close to each other in terms of path edges between them in the same cluster; and (2) vertices belonging to different clusters are relatively distant from each other in terms of clustered path edges between them.

Figure 4.3 gives an illustrative example of two vertex-centric path graphs about authors. For A-P-A path graph in Figure 4.3 (a), the number associated with an author vertex represents the number of coauthored papers by this author. Here, we only consider coauthored papers on three DB conferences: *SIGMOD*, *VLDB*, *ICDE* and three DM conferences: *KDD*, *ICDM*, *SDM*. For A-P-V-P-A meta path graph in Figure 4.3 (b), the number associated

to an author, e.g., *Philip S. Yu* (199), represents the total number of papers published by this author on the above six venues. Similarly, the number on a path edge specifies the value of this path edge through link composition by multiplying adjacency matrices, e.g., $\mathbf{W}_{AP} \times \mathbf{W}_{PA}$ in Figure 4.3 (a), and $\mathbf{W}_{AP} \times \mathbf{W}_{PV} \times \mathbf{W}_{VP} \times \mathbf{W}_{PA}$) in Figure 4.3 (b).

4.4 The VEPathCluster Approach

VEPathCluster improves the clustering quality by utilizing four novel mining strategies: (1) edge-centric random walk model; (2) clustering-based multigraph model; (3) integration of vertex-centric clustering and edge-centric clustering; and (4) dynamic weight learning. VEPathCluster iteratively performs the following three tasks to achieve high quality clustering: (1) fix edge clustering and weight assignment to update vertex clustering; (2) fix vertex clustering and weight assignment to update edge clustering; and (3) fix vertex clustering and edge clustering to update weight assignment.

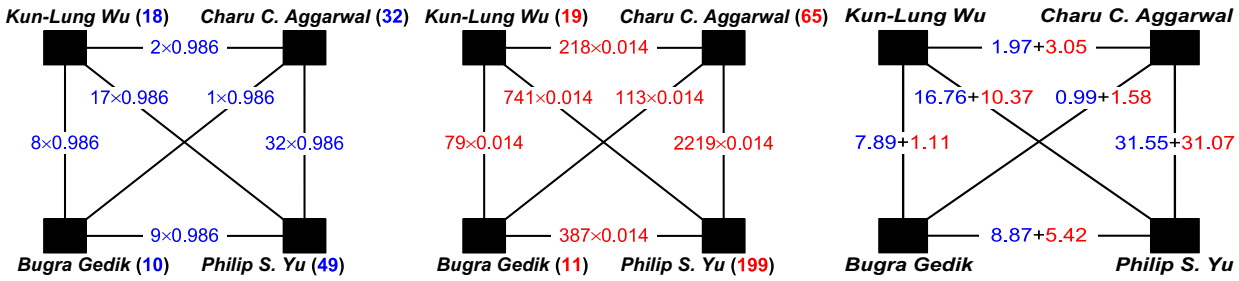
4.4.1 Initialization

Given a heterogeneous network $G = (V, E)$, the set of target vertices $V_c \subset V$, and the M target meta paths, the number of clusters K, we first construct the M vertex-centric path graphs: $VG_1, \cdots, VG_M$. Then we initialize the weight assignment and produce the initial vertex clustering of V_c on K clusters.

Let $\omega_m^{(1)}$ $(1 \leq m \leq M)$ be the weight for the m^{th} vertex-centric path graph VG_m at the first iteration, and $\mathbf{P}_m$ be the adjacency matrix of VG_m. We use the initial weights $\omega_1^{(t)}, \cdots, \omega_M^{(t)}$ to integrate M vertex-centric path graphs into a unified vertex-centric path graph VG. The matrix form of VG, denoted by $\mathbf{P}^{(1)}$, is defined below.

$$\mathbf{P}^{(1)} = \omega_1^{(1)}\mathbf{P}_1 + \cdots + \omega_M^{(1)}\mathbf{P}_M \; s.t. \; \sum_{m=1}^{M} \omega_m^{(1)} = 1, \; \omega_1^{(1)}, \cdots, \omega_M^{(1)} \geqslant 0 \tag{4.1}$$

Random weight assignment often performs poorly and results in incorrect clustering



(a) A-P-A Path Graph (b) A-P-V-P-A Path Graph (c) Unified Path Graph

Figure 4.4: Unified Vertex-centric Path Graph

results due to the sharp difference in edge values from path graph to path graph, e.g., the edge values in Figure 4.3 (a) are between 1 and 32 but the edge values in Figure 4.3 (b) are between 79 and 2219. We normalize edge values in each VG_m by assigning an initial weight for each VG_m in terms of its maximal edge value, i.e., $\omega_1^{(1)} = \frac{1/\max \mathbf{P}_1}{\sum_{m=1}^{M} 1/\max \mathbf{P}_m}, \ldots, \omega_M^{(1)} = \frac{1/\max \mathbf{P}_M}{\sum_{m=1}^{M} 1/\max \mathbf{P}_m}$, where $\max \mathbf{P}_m$ represents the maximal element in $\mathbf{P}_m$.

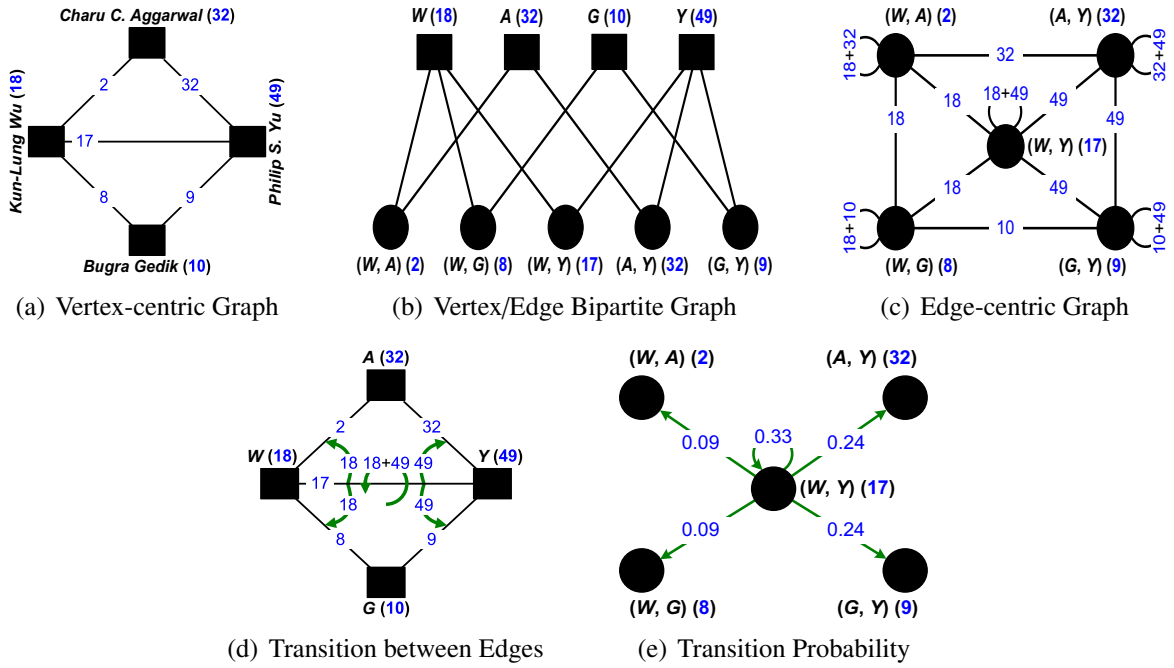
For two path graphs in Figure 4.3, we multiply the edge values by the initial weights $\frac{1/32}{1/32+1/2219} = 0.986$ and $\frac{1/2219}{1/32+1/2219} = 0.014$ to generate two path graphs in Figures 4.4 (a) and (b). Figure 4.4 (c) shows the combination of them with the above initial weights.

Next we employ a soft clustering method, Fuzzy C-Means (FCM) [67], on the unified vertex-centric path graph VG, to cluster each vertex to K clusters such that it has up to K membership probabilities. We use symbol $\mathbf{X}_k^{(1)}(i)$ to represent the membership probability of a vertex $v_i \in V_c$ $(1 \leq i \leq N_{V_c})$ belonging to cluster c_k $(1 \leq k \leq K)$ at the first iteration. Figure 4.6 (a) exhibits the FCM clustering result of author vertices in Figure 4.4 (c), where each green number and ochre number in the bracket denotes the membership probability of an author belonging to cluster *DB* or *DM* respectively.

4.4.2 Edge-centric Random Walk Model

Edge-centric random walk model is constructed by performing two tasks: (1) for each vertex-centric path graph, construct an edge-centric path graph and define its vertex values and edge values; and (2) define the transition probability on the edge-centric path graph.

Let VG_m be a vertex-centric path graph corresponding to the m^{th} meta path MP_m. We



(a) Vertex-centric Graph (b) Vertex/Edge Bipartite Graph (c) Edge-centric Graph

(d) Transition between Edges (e) Transition Probability

Figure 4.5: Random Walk on Edges

build an edge-centric path graph EG_m by converting the edges and vertices of VG_m to the vertices and edges of EG_m respectively. For example, we first transform the vertex-centric graph in Figure 4.5 (a) into a vertex/edge bipartite graph in Figure 4.5 (b) where rectangle vertices and circle vertices correspond to the vertices and the edges in Figure 4.5 (a). The circle vertex (W, Y) (17) in Figure 4.5 (b) corresponds to the edge between *Kun-Lung* ***Wu*** and *Philip* ***Yu*** with weight of 17 in Figure 4.5 (a).

Next we convert the bipartite graph in Figure 4.5 (b) to the edge-centric graph in Figure 4.5 (c) by shrinking each common rectangle vertex shared by any pair of circle vertices to an edge between these two circle vertices, and assign the edge value with the value of the common rectangle vertex in Figure 4.5 (b). For instance, a common rectangle vertex W (18) shared by two circle vertices (W, Y) (17) and (W, G) (8) in Figure 4.5 (b) is converted to the edge between (W, Y) (17) and (W, G) (8) in Figure 4.5 (c). In addition, to capture the fact that a circle vertex connects to two rectangle vertices in Figure 4.5 (b), we build a spin edge for each circle vertex in Figure 4.5 (c). The value of this spin edge is the sum of the values of two rectangle vertices linked to this circle vertex in the bipartite graph.

We define the transition probability on EG_m such that the edge-centric random walk model can be employed to measure the closeness between a pair of edge vertices in EG_m.

Definition 15 *[Transition Probability on Edge-centric Path Graph] Let $VG_m = (V_c, E_m)$ be a vertex-centric path graph where V_c is the set of target vertices, E is the set of path edges between vertices in V_c and $EG_m = (E_m, E_m \times E_m)$ is a corresponding edge-centric path graph. The transition probability on EG_m is defined below.*

$$\mathbf{T}_m(e_{mi}, e_{mj}) = \begin{cases} \dfrac{\mathbf{Q}_m(e_{mi}, e_{mj})}{\sum_{l=1}^{N_{E_m}} \mathbf{Q}_m(e_{ml}, e_{mj})}, (e_{mi}, e_{mj}) \in E_m \times E_m, \\ 0, \qquad\qquad otherwise. \end{cases}, \ 1 \le m \le M \tag{4.2}$$

where $\mathbf{Q}_m$ is the adjacency matrix of EG_m and $\mathbf{T}_m(e_{mi}, e_{mj})$ represents the transition probability from vertex e_{mi} to vertex e_{mj} in EG_m.

Consider Figure 4.5, we compute the transition probabilities from (W, Y) to all five circle vertices: Given that $\sum_{l=1}^{N_{E_m}} \mathbf{Q}_m(e_{ml}, e_{mj}) = (18 + 49) + 18 + 18 + 49 + 49 = 201$, the transition probability from (W, Y) to (W, G) is $18/201 = 0.09$.

We express the above transition probability in a matrix form.

$$\mathbf{T}_m = \mathbf{Q}_m \mathbf{D}^{-1}, \ 1 \le m \le M \tag{4.3}$$

where $\mathbf{D}$ is a diagonal matrix $\mathbf{D} = diag(d_1, \cdots, d_{N_{E_m}})$ and $d_j = \sum_{l=1}^{N_{E_m}} \mathbf{Q}_m(e_{ml}, e_{mj})$ $(1 \le j \le N_{E_m})$.

4.4.3 Clustering-based Multigraph Model

The second novelty is to perform clustering analysis on vertex-centric multigraph and edge-centric multigraph to effectively combine vertex homophily with edge homophily. Recall Figure 4.2 (b), assigning *Kun-Lung Wu* to *Philip S. Yu* is due to the using of aggregated edge weight (i.e., the total number of coauthored papers) to measure the vertex closeness. We address this problem by introducing two clustering-based multigraph models, one for

vertex-centric path graphs and another for edge-centric path graphs.

Given that a vertex-centric path graph $VG_m = (V_c, E_m)$, and the clustering result on the corresponding edge-centric path graph $EG_m = (E_m, E_m \times E_m)$ obtained at the previous iteration. A vertex-centric path multigraph i.e., $\mathbf{Y}_m^{(t-1)}$, denoted as $VMG_m = (V_c, F_m)$, is an edge augmented multigraph, where F_m is the set of edges satisfying the following condition: for each edge $(v_i, v_j) \in E_m$ in VG_m, we create a set of parallel edges between v_i and v_j in F_m. Each set of edges has up to K clustered edges and each of the parallel edges corresponds to a certain cluster c_k. The value of the parallel edge with label c_k between v_i and v_j in VMG_m at the t^{th} iteration, denoted by $\mathbf{P}_{mk}^{(t)}(v_i, v_j)$, are computed as follow.

$$\mathbf{P}_{mk}^{(t)}(v_i, v_j) = \mathbf{P}_m(v_i, v_j) \times \mathbf{Y}_{mk}^{(t-1)}((v_i, v_j)),\ 1 \le m \le M,\ 1 \le k \le K \tag{4.4}$$

where $\mathbf{P}_m(v_i, v_j)$ represents the value of the edge between v_i and v_j in VG_m. $\mathbf{Y}_{mk}^{(t-1)}$ denotes the k^{th} column vector of the edge clustering membership matrix $\mathbf{Y}_m^{(t-1)}$ and $\mathbf{Y}_{mk}^{(t-1)}((v_i, v_j))$ specifies the membership probability of vertex (v_i, v_j) belonging to cluster c_k in EG_m at the last iteration. $\mathbf{P}_{mk}^{(t)}$ is essentially a projection of $\mathbf{P}_m$ on c_k.

Similarly, let $\mathbf{X}^{(t)}$ $(t \ge 1)$ be the soft clustering result on the unified vertex-centric path multigraph VMG at the current iteration. For each edge-centric path graph $EG_m = (E_m, E_m \times E_m)$, we create an edge-centric path multigraph EMG_m: for each edge $(e_{mi}, e_{mj}) \in E_m \times E_m$, we create a set of up to K parallel edges. Each of parallel edges corresponds to cluster c_k. The edge values on EMG_m at the t^{th} iteration are defined as follow.

$$\mathbf{Q}_{mk}^{(t)}(e_{mi}, e_{mj}) = \begin{cases} \mathbf{Q}_m(e_{mi}, e_{mj}) \times \mathbf{X}_k^{(t)}(e_{mi} \wedge e_{mj}), & e_{mi} \ne e_{mj}, \\ \mathbf{R}_m(v_a) \times \mathbf{X}_k^{(t)}(v_a) + \mathbf{R}_m(v_b) \times \mathbf{X}_k^{(t)}(v_b), & e_{mi} = e_{mj}. \end{cases}, \tag{4.5}$$
$$1 \le m \le M,\ 1 \le k \le K$$

where $\mathbf{Q}_m(e_{mi}, e_{mj})$ specifies the edge value between two vertices e_{mi} and e_{mj} in EG_m, $\mathbf{X}_k^{(t)}$

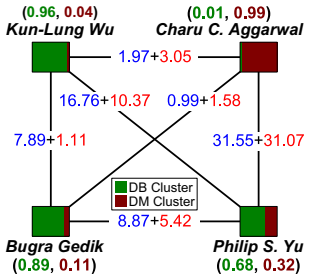
denotes the k^{th} column vector of the vertex clustering membership matrix $\mathbf{X}^{(t)}$ and $\mathbf{X}_k^{(t)}((e_{mi} \wedge e_{mj}))$ specifies the membership probability of common vertex of two edges e_{mi} and e_{mj} belonging to cluster c_k in the unified vertex-centric path graph VG at the t^{th} iteration. $\mathbf{Q}_{mk}^{(t)}$ is essentially a projection of $\mathbf{Q}_m$ on cluster c_k. When $e_{mi} = e_{mj}$, edge (e_{mi}, e_{mj}) is a spin edge associated to e_{mi} in EG_m. In this situation, e_{mi} and e_{mj} correspond to the same edge in VG_m, and e_{mi} and e_{mj} will have the same two endpoints (v_a and v_b) in VG_m, e.g., the spin edge $((W, Y), (W, Y))$ in Figure 4.5 (b) and the edge between *Kun-Lung Wu* and *Philip S. Yu* in Figure 4.5 (a). $\mathbf{R}_m(v_x)$ represents the value of endpoint v_x in VG_m, say 18 for *Kun-Lung Wu* in Figure 4.5 (a), and $\mathbf{X}_k^{(t)}(v_x)$ denotes the probability of v_x belonging to c_k in VG or VMG at the t^{th} iteration.

For ease of presentation, we omit all spin edges in Figure 4.6. Based on the A-P-A edge-centric path graph in Figure 4.6 (b) and its vertex soft clustering result in Figure 4.6 (a), we generate the A-P-A edge-centric path multigraph in Figure 4.6 (c). Using the probabilities of *Kun-Lung Wu* on clusters *DB* and *DM*: (0.96, 0.04) in Figure 4.6 (a) and the edge between (W, Y) and (W, A) in Figure 4.6 (b), we produce two parallel edges between (W, Y) and (W, A) in Figure 4.6 (c) as $18 \times 0.96 = 17.28$ and $18 \times 0.04 = 0.72$ respectively.

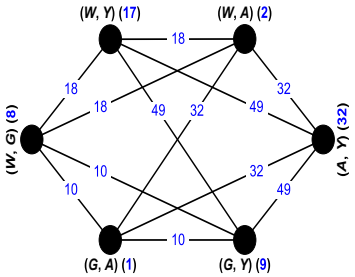
4.4.4 Edge-centric Clustering

We perform edge-centric soft clustering in two steps: (1) convert each edge-centric path graph EG_m to an edge-centric path multigraph EMG_m based on the vertex soft clustering $\mathbf{X}^{(1)}$ on the unified vertex-centric path graph VG or $\mathbf{X}^{(t)}$ ($t > 1$) on the unified vertex-centric path multigraph VMG; and (2) compute the edge soft clustering $\mathbf{Y}_m^{(t)}$ on each edge-centric path multigraph EMG_m.

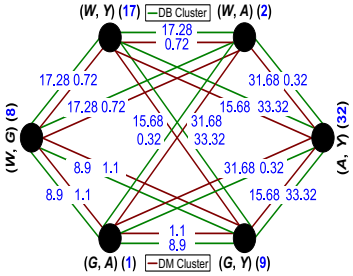
Different from traditional unsupervised graph clustering methods, at the first clustering iteration, we adopt a semi-supervised manner on each EG_m with the geometric mean of the probabilities of two endpoints belonging to cluster c_k as the initial membership probability of an edge on c_k. This is motivated by the observation that if the membership probabilities



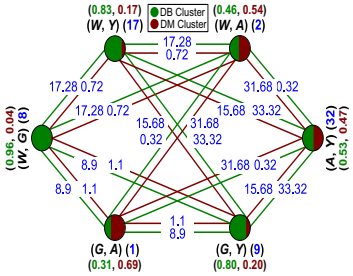
(a) Vertex Clustering w.r.t. Fig. 4.4 (c)



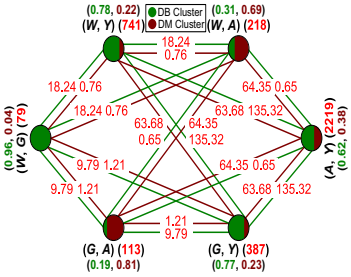
(b) A-P-A Edge-centric Path Graph w.r.t. Fig. 4.3 (a)



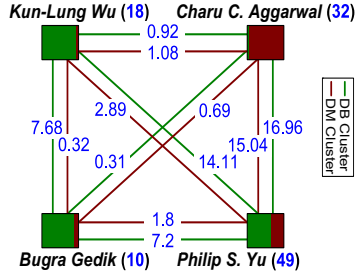
(c) A-P-A Edge-centric Path Multigraph w.r.t. Fig. 4.6 (a) + Fig. 4.6 (b)



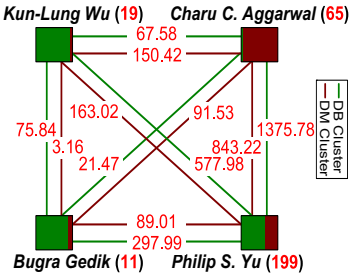
(d) A-P-A Edge Clustering w.r.t. Fig. 4.6 (a) + Fig. 4.6 (c)



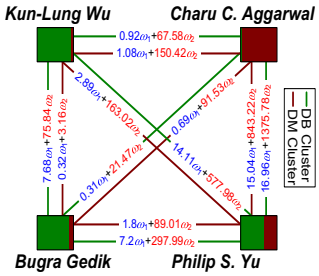
(e) A-P-V-P-A Edge Clustering w.r.t. Fig. 4.3 (b) + Fig. 4.6 (a)



(f) A-P-A Vertex-centric Path Multigraph w.r.t. Fig. 4.3 (a) + Fig. 4.6 (d)



(g) A-P-V-P-A Vertex-centric Path Multigraph w.r.t. Fig. 4.3 (b) + Fig. 4.6 (e)



(h) Unified Vertex-centric Path Multigraph w.r.t. Fig. 4.6 (f) + Fig. 4.6 (g)

Figure 4.6: Iterative Vertex Clustering on Vertex-centric Path Multigraph and Edge Clustering on Edge-centric Path Multigraph

of two associated endpoints of an edge belonging to c_k are very large, then it is highly probable that this edge also has a large probability on c_k.

Formally, we convert each EG_m to an EMG_m by converting the adjacency matrix of EG_m to up to K independent adjacency matrices in terms of the cluster labels of the edges in EG_m, and then learns the cluster probabilities of edge vertices in EG_m on c_k based on the k^{th} adjacency matrix. Let (v_i, v_j) be an edge vertex in EG_m where v_i and v_j are the target vertices in the corresponding $VG_m = (V_c, E_m)$, and $\mathbf{X}_k^{(1)}(v_x)$ be the cluster membership probability of $v_x \in V_c$ belonging to cluster c_k at the first iteration. We define the initial edge clustering membership matrix $\mathbf{Y}_m^{(0)}$ for EMG_m below.

$$\mathbf{Y}_{mk}^{(0)}((v_i, v_j)) = \frac{\sqrt{\mathbf{X}_k^{(1)}(v_i) \times \mathbf{X}_k^{(1)}(v_j)}}{\sum_{l=1}^{K} \sqrt{\mathbf{X}_l^{(1)}(v_i) \times \mathbf{X}_l^{(1)}(v_j)}},\ 1 \le m \le M,\ 1 \le k \le K \tag{4.6}$$

where $\mathbf{Y}_{mk}^{(0)}$ is the k^{th} column vector of $\mathbf{Y}_m^{(0)}$, $\mathbf{Y}_{mk}^{(0)}((v_i, v_j))$ represents the initial membership probability of edge vertex (v_i, v_j) on c_k in EMG_m, and $\mathbf{X}_k^{(1)}(v_x)$ specifies the probability of v_x on c_k in VG.

Based on the initial vertex clustering membership matrix $\mathbf{X}^{(1)}$ for VG or the vertex clustering membership matrix $\mathbf{X}^{(t)}$ $(t > 1)$ for VMG, we transform each EG_m into an edge-centric path multigraph EMG_m by Eq. (4.5). In the first clustering iteration, we update $\mathbf{Y}_m^{(1)}$ with $\mathbf{Y}_m^{(0)}$ based on $\mathbf{X}^{(1)}$ for VG through label propagation and update $\mathbf{Y}_m^{(t)}$ with $\mathbf{Y}_m^{(t-1)}$ in each subsequent iteration t $(t > 1)$.

Similar to Eq.(4.2), the transition probability on each EMG_m at the current iteration is defined by normalizing each kind of parallel edges with the same cluster labels in EMG_m

as follow.

$$\mathbf{T}_{mk}^{(t)}(e_{mi}, e_{mj}) = \begin{cases} \dfrac{\mathbf{Q}_{mk}^{(t)}(e_{mi}, e_{mj})}{\sum_{l=1}^{N_{E_m}} \mathbf{Q}_{mk}^{(t)}(e_{ml}, e_{mj})}, \mathbf{Q}_{mk}^{(t)}(e_{ml}, e_{mj}) \neq 0, \\ 0, \qquad\qquad otherwise. \end{cases}, \\ 1 \leq m \leq M,\ 1 \leq k \leq K \tag{4.7}$$

where $\mathbf{T}_{mk}^{(t)}(e_{mi}, e_{mj})$ denotes the transition probability with cluster label c_k on one of parallel edges between edge vertices e_{mi} and e_{mj} in EMG_m. The transition matrix on EMG_m is given below.

$$\mathbf{T}_{mk}^{(t)} = \mathbf{Q}_{mk}^{(t)}(\mathbf{D}_{mk}^{-1})^{(t)},\ 1 \leq m \leq M,\ 1 \leq k \leq K \tag{4.8}$$

where $(\mathbf{D}_{mk}^{-1})^{(t)}$ is a diagonal matrix $(\mathbf{D}_{mk}^{-1})^{(t)} = \mathrm{diag}(d_1, \cdots, d_{N_{E_m}})$, and $d_j = \sum_{l=1}^{N_{E_m}} \mathbf{Q}_{mk}^{(t)}(e_{ml}, e_{mj})$ $(1 \leq j \leq N_{E_m})$.

Thus, we produce K edge clustering kernels $\mathbf{T}_{mk}^{(t)}$, each corresponding to cluster c_k $(1 \leq k \leq K)$. The transition operation in each edge-centric path multigraph is divided into two steps: (1) choose those parallel edges with the objective cluster label by clustering objective; and (2) select an edge with the largest probability from the above edges to jump.

Let $\mathbf{Y}_m = [\mathbf{Y}_{m1}, \mathbf{Y}_{m2}, \cdots, \mathbf{Y}_{mK}] \in \mathbb{R}^{N_{E_m} \times K}$ be the edge clustering membership matrix for E_m in EMG_m $(1 \leq m \leq M)$. For each edge clustering membership vector $\mathbf{Y}_{mk}$ $(1 \leq k \leq K)$ based on cluster c_k, we use an individual clustering kernel $\mathbf{T}_{mk}^{(t)}$ to iteratively infer the membership probabilities of all edge vertices in E_m on c_k.

$$\begin{aligned} &Initilization:\ \mathbf{Y}_{mk} = \mathbf{Y}_{mk}^{(t-1)} \\ &Iteration:\ \mathbf{Y}_{mk} = \mathbf{T}_{mk}^{(t)} \mathbf{Y}_{mk} \end{aligned} \tag{4.9}$$

Based on the edge clustering membership matrix $\mathbf{Y}_{mk}^{(t-1)}$ at the last clustering round, VEPathCluster iteratively infers the membership probabilities of vertices in E_m until $\mathbf{Y}_{mk}$ converges. We then normalize each entry $\mathbf{Y}_{mk}(e_{mi})$ $(1 \leq i \leq N_{E_m})$ in $\mathbf{Y}_{mk}$ as follow.

$$\mathbf{Y}_{mk}^{(t)}(e_{mi}) = \frac{\mathbf{Y}_{mk}(e_{mi})}{\sum_{l=1}^{K} \mathbf{Y}_{ml}(e_{mi})} \tag{4.10}$$

where $e_{mi} \in E_m$ represents an edge vertex in EMG_m and $\mathbf{Y}_{mk}^{(t)}$ specifies the normalized edge clustering membership vector based on c_k. Thus, the edge clustering membership matrix is updated below.

$$\mathbf{Y}_{m}^{(t)} = \left[\mathbf{Y}_{m1}^{(t)} \quad \mathbf{Y}_{m2}^{(t)} \quad \cdots \quad \mathbf{Y}_{mK}^{(t)}\right],\ 1 \leq m \leq M \tag{4.11}$$

For example, based on the vertex clustering in Figure 4.6 (a) and the edge-centric path multigraph in Figure 4.6 (c), we produce the A-P-A edge clustering in Figure 4.6 (d).

4.4.5 Vertex-centric Clustering

The vertex clustering on the unified vertex-centric path multigraph VMG follows the heuristic rule: if vertex $v_i \in V_c$ in each vertex-centric path graph VG_m has many neighbors with large probabilities on cluster c_k and the edges between v_i and these neighbors have large probabilities on c_k, then it is highly probable that v_i belongs to c_k with a larger probability. In each iteration, we use the edge clustering result on each edge-centric path graph EG_m at the previous iteration ($\mathbf{Y}_m^{(t-1)}$) to perform the vertex clustering on VMG at the current iteration ($\mathbf{X}^{(t)}$) in three steps.

(1) Based on $\mathbf{Y}_m^{(t-1)}$ and Eq.(4.4), we first convert each VG_m to an vertex-centric path multigraph VMG_m by transforming the adjacency matrix of VG_m into K independent adjacency matrices in terms of the cluster labels of parallel edges. For example, based on the edge clustering result on the edge-centric path multigraph in Figure 4.6 (d) (or Figure 4.6 (e)), we convert the vertex-centric path graph in Figure 4.3 (a) (or Figure 4.3 (b)) to the

vertex-centric path multigraph in Figure 4.6 (f) (or Figure 4.6 (g)).

(2) We combining M vertex-centric path multigraphs VMG_m into the unified vertex-centric path multigraph VMG based on each of K edge clusters with weighting factors $\omega_1^{(t)}$, $\cdots$, $\omega_N^{(t)}$. A dynamic weight tuning mechanism will be detailed in Section 4.4.6. Thus, we compute the value of the unified parallel edge between vertices v_i and v_j in VMG about cluster c_k at the t^{th} iteration as follow.

$$\begin{aligned} &\mathbf{P}_k^{(t)}(v_i, v_j) = \omega_1^{(t)}\mathbf{P}_{1k}^{(t)}(v_i, v_j) + \cdots + \omega_M^{(t)}\mathbf{P}_{Mk}^{(t)}(v_i, v_j),\ 1 \leq k \leq K \\ &s.t.\ \sum_{m=1}^{M} \omega_m^{(t)} = 1,\ \omega_1^{(t)}, \cdots, \omega_M^{(t)} \geqslant 0 \end{aligned} \tag{4.12}$$

where $\omega_m^{(t)}$ $(1 \leq m \leq M)$ represents the weight for the m^{th} vertex-centric path multigraph VMG_m at the t^{th} iteration, and $\mathbf{P}_{mk}^{(t)}(v_i, v_j)$ specifies the value of the parallel edge with label c_k between v_i and v_j in VMG_m. Note that $\mathbf{P}_k^{(t)}(v_i, v_j)$ keeps changing with $\omega_1^{(t)}$, $\cdots$, $\omega_M^{(t)}$ through dynamic weight learning during each iteration.

The matrix form of VMG is defined based on K kinds of clustered parallel edges.

$$\begin{aligned} &\mathbf{P}_1^{(t)} = \omega_1^{(t)}\mathbf{P}_{11}^{(t)} + \omega_2^{(t)}\mathbf{P}_{21}^{(t)} + \cdots + \omega_M^{(t)}\mathbf{P}_{M1}^{(t)} \\ &\qquad\qquad \cdots \qquad\qquad , \\ &\mathbf{P}_K^{(t)} = \omega_1^{(t)}\mathbf{P}_{1K}^{(t)} + \omega_2^{(t)}\mathbf{P}_{2K}^{(t)} + \cdots + \omega_M^{(t)}\mathbf{P}_{MK}^{(t)} \\ &s.t.\ \sum_{m=1}^{M} \omega_m^{(t)} = 1,\ \omega_1^{(t)}, \cdots, \omega_M^{(t)} \geqslant 0 \end{aligned} \tag{4.13}$$

Figure 4.6 (h) shows the unified vertex-centric path multigraph by combining the two vertex-centric path multigraphs in Figure 4.6 (f) and (g) with the weights of ω_1 and ω_2 respectively such that the clustered path edges with the same labels between the same pair of vertices from two vertex-centric path multigraphs are combined.

(3) We compute the vertex clustering membership matrix $\mathbf{X} = [\mathbf{X}_1, \mathbf{X}_2, \cdots, \mathbf{X}_K] \in \mathbb{R}^{N_{V_c} \times K}$ for the target vertices V_c in VMG. We below define the transition probability on VMG in terms of each of K edge clusters.

$$\mathbf{S}_k^{(t)}(v_i, v_j) = \begin{cases} \dfrac{\mathbf{P}_k^{(t)}(v_i, v_j)}{\sum_{l=1}^{N_{V_c}} \mathbf{P}_k^{(t)}(v_l, v_j)}, \mathbf{P}_k^{(t)}(v_i, v_j) \neq 0, \\ 0, \qquad\qquad otherwise. \end{cases}, \ 1 \leq k \leq K \tag{4.14}$$

where $\mathbf{S}_k^{(t)}(v_i, v_j)$ denotes the transition probability with cluster label c_k on one of parallel edges between vertex v_i and vertex v_j in VMG.

The transition matrix on VMG is given as follow.

$$\mathbf{S}_k^{(t)} = \mathbf{P}_k^{(t)} (\mathbf{D}_k^{-1})^{(t)}, \ 1 \leq k \leq K \tag{4.15}$$

where $(\mathbf{D}_k^{-1})^{(t)}$ is a diagonal matrix $(\mathbf{D}_k^{-1})^{(t)} = \text{diag}(d_1, \cdots, d_{N_{V_c}})$, and $d_j = \sum_{l=1}^{N_{V_c}} \mathbf{P}_k^{(t)}(v_l, v_j)$ $(1 \leq j \leq N_{V_c})$.

Similar to edge-centric clustering, we produce K vertex clustering kernels $\mathbf{S}_k^{(t)}$, each corresponding to cluster c_k. The transition operation in the unified vertex-centric path multigraph VMG is divided into two steps: (1) choose those parallel edges with the objective cluster label; and (2) select an edge with the largest probability from the above edges to move.

For each vertex clustering membership vector $\mathbf{X}_k$ $(1 \leq k \leq K)$ based on c_k, we utilize an individual clustering kernel $\mathbf{S}_k^{(t)}$ to iteratively infer the membership probabilities of vertices in V_c on c_k.

$$\begin{aligned} &Initilization: \ \mathbf{X}_k = \mathbf{X}_k^{(t-1)} \\ &Iteration: \ \mathbf{X}_k = \mathbf{S}_k^{(t)} \mathbf{X}_k \end{aligned} \tag{4.16}$$

When the iterative vertex clustering converges, we further normalize each entry $\mathbf{X}_k(v_i)$

$(1 \leq i \leq N_{V_c})$ in $\mathbf{X}_k$ $(1 \leq k \leq K)$ below.

$$\mathbf{X}_k^{(t)}(v_i) = \frac{\mathbf{X}_k(v_i)}{\sum_{l=1}^{K} \mathbf{X}_l(v_i)} \tag{4.17}$$

where $v_i \in V_c$ denotes a target vertex in VMG and $\mathbf{X}_k^{(t)}$ represents the normalized vertex clustering membership vector based on c_k. Thus, the vertex clustering membership matrix is updated below.

$$\mathbf{X}^{(t)} = \begin{bmatrix} \mathbf{X}_1^{(t)} & \mathbf{X}_2^{(t)} & \cdots & \mathbf{X}_K^{(t)} \end{bmatrix} \tag{4.18}$$

$\mathbf{X}^{(t)}$ will be used to enter the next vertex clustering round.

4.4.6 Clustering with Weight Learning

The objective function of VEPathCluster is defined to maximize fuzzy intra-cluster similarity [68, 69] for both vertex clustering in the unified vertex-centric path multigraph VMG and edge clustering on each edge-centric path multigraph EMG_m.

Definition 16 *[VEPathCluster Clustering Objective Function] Let VMG be a unified vertex-centric path multigraph, VMG_m $(m \in \{1, \cdots, M\})$ be M vertex-centric path multigraphs, EMG_m $(m \in \{1, \cdots, M\})$ be M edge-centric path multigraphs, $\omega_1, \cdots, \omega_M$ be the weighting factors for $VMG_1, \cdots,$*

VMG_M and $EMG_1, \cdots, EMG_M$ defined in Eqs.(4.12) and (4.13) respectively, given K vertex soft clusters for VMG with a membership matrix $\mathbf{X}$ and K path edge soft clusters for each EMG_m with a membership matrix $\mathbf{Y}_m$, the goal of VEPathCluster *is to maximize the*

following objective function.

$$
\begin{aligned}
O(\mathbf{X}, \mathbf{Y}_1, \cdots, \mathbf{Y}_M, \omega_1, \cdots, \omega_M) = & \sum_{i=1}^{N_{V_c}} \sum_{j=1}^{N_{V_c}} \sum_{k=1}^{K} \mathbf{X}_k(v_i)\mathbf{X}_k(v_j)\mathbf{P}_k(v_i, v_j) \\
& + \sum_{m=1}^{M} \sum_{i=1}^{N_{E_m}} \sum_{j=1}^{N_{E_m}} \sum_{k=1}^{K} \mathbf{Y}_{mk}(e_{mi})\mathbf{Y}_{mk}(e_{mj})\mathbf{Q}_{mk}(e_{mi}, e_{mj}) \qquad (4.19) \\
\max_{\omega_1, \cdots, \omega_M} O(\mathbf{X}, \mathbf{Y}_1, \cdots, \mathbf{Y}_M, \omega_1, \cdots, \omega_M), \ & s.t. \ \sum_{m=1}^{M} \omega_m = 1, \ \omega_1, \cdots, \omega_M \geqslant 0
\end{aligned}
$$

According to Eqs.(4.4)-(4.18), the objective function O is a fractional function of multi variables $\omega_1, \cdots, \omega_M$ with non-negative real coefficients. On the other hand, the numerator and the denominator of O are both polynomial functions of the above variables. Without loss of generality, we rewrite Eq.(4.19) as follow.

$$
\begin{aligned}
\max_{\omega_1, \cdots, \omega_M} O(\mathbf{X}, \mathbf{Y}_1, \cdots, \mathbf{Y}_M, \omega_1, \cdots, \omega_M) = & \max_{\omega_1, \cdots, \omega_M} \frac{\sum_{i=1}^{p} a_i \prod_{j=1}^{M} (\omega_j)^{b_{ij}}}{\sum_{i=1}^{q} o_i \prod_{j=1}^{M} (\omega_j)^{r_{ij}}} \\
a_i, b_{ij}, o_i, r_{ij} \geq 0, b_{ij}, r_{ij} \in \mathbb{Z}, \ & s.t. \ \sum_{m=1}^{M} \omega_m = 1, \ \omega_1, \cdots, \omega_M \geqslant 0
\end{aligned} \qquad (4.20)
$$

where there are p polynomial terms in the numerator and q polynomial terms in the denominator, a_i and o_i are the coefficients of the i^{th} terms respectively, and b_{ij} and r_{ij} are the exponents of corresponding variables in the i^{th} terms respectively.

For ease of presentation, we revise the original objective as the following nonlinear fractional programming problem (NFPP).

Definition 17 *[Nonlinear Fractional Programming Problem] Let* $f(\omega_1, \cdots, \omega_M) = \sum_{i=1}^{p} a_i \prod_{j=1}^{M} (\omega_j)^{b_{ij}}$ *and* $g(\omega_1, \cdots, \omega_M) = \sum_{i=1}^{q} o_i \prod_{j=1}^{M} (\omega_j)^{r_{ij}}$*, the clustering goal is revised as fol-*

low.

$$\max_{\omega_1,\cdots,\omega_M} \frac{f(\omega_1,\cdots,\omega_M)}{g(\omega_1,\cdots,\omega_M)},\ s.t.\ \sum_{m=1}^{M}\omega_m = 1,\ \omega_1,\cdots,\omega_M \geqslant 0 \tag{4.21}$$

Our clustering objective is equivalent to maximize a quotient of two polynomial functions of multiple variables. It is very hard to perform function trend identification and estimation to determine the existence and uniqueness of solutions. Therefore, we want to transform this sophisticated NFPP into an easily solvable problem.

Definition 18 *[Nonlinear Parametric Programming Problem] Let* $f(\omega_1,\cdots,\omega_M) = \sum_{i=1}^{p} a_i \prod_{j=1}^{M}(\omega_j)^{b_{ij}}$ *and* $g(\omega_1,\cdots,\omega_M) = \sum_{i=1}^{q} o_i \prod_{j=1}^{M}(\omega_j)^{r_{ij}}$*, the NPPP is defined as follow.*

$$\begin{aligned} F(\gamma) = &\max_{\omega_1,\cdots,\omega_M} f(\omega_1,\cdots,\omega_M) - \gamma g(\omega_1,\cdots,\omega_M), \\ &s.t.\ \sum_{m=1}^{M}\omega_m = 1,\ \omega_1,\cdots,\omega_M \geqslant 0 \end{aligned} \tag{4.22}$$

Theorem 14 *The NFPP in Definition 17 is equivalent to the NPPP in Definition 18, i.e.,* $\gamma = \max_{\omega_1,\cdots,\omega_M} \frac{f(\omega_1,\cdots,\omega_M)}{g(\omega_1,\cdots,\omega_M)}$ *if and only if* $F(\gamma) = \max_{\omega_1,\cdots,\omega_M} f(\omega_1,\cdots,\omega_M) - \gamma g(\omega_1,\cdots,\omega_M) = 0$.

Proof. If $(\overline{\omega}_1,\cdots,\overline{\omega}_M)$ *is a feasible solution of* $F(\gamma) = 0$*, then* $f(\overline{\omega}_1,\cdots,\overline{\omega}_M) - \gamma g(\overline{\omega}_1,\cdots,\overline{\omega}_M) = 0$*. Thus* $f(\omega_1,\cdots,\omega_M) - \gamma g(\omega_1,\cdots,\omega_M) \leqslant f(\overline{\omega}_1,\cdots,\overline{\omega}_M) - \gamma g(\overline{\omega}_1,\cdots,\overline{\omega}_M) = 0$*. We have* $\gamma = f(\overline{\omega}_1,\cdots,\overline{\omega}_M)/g(\overline{\omega}_1,\cdots,\overline{\omega}_M) \geqslant f(\omega_1,\cdots,\omega_M)/g(\omega_1,\cdots,\omega_M)$*. Thus* γ *is a maximum value of NFPP and* $(\overline{\omega}_1,\cdots,\overline{\omega}_M)$ *is an optimal solution of NFPP.*

Conversely, if $(\omega_1,\cdots,\overline{\omega}_M)$ *solves NFPP, then we have* $\gamma = f(\overline{\omega}_1,\cdots,\overline{\omega}_M)/g(\overline{\omega}_1,\cdots,\overline{\omega}_M) \geqslant f(\omega_1,\cdots,\omega_M)/g(\omega_1,\cdots,\omega_M)$*. Thus* $f(\omega_1,\cdots,\omega_M) - \gamma g(\omega_1,\cdots,\omega_M) \leqslant f(\overline{\omega}_1,\cdots,\overline{\omega}_M) - \gamma g(\overline{\omega}_1,\cdots,\overline{\omega}_M) = 0$*. We have* $F(\gamma) = 0$ *and the maximum is taken at* $(\overline{\omega}_1,\cdots,\overline{\omega}_M)$*.*

Now the original NFPP has been successfully transformed into the straightforward NPPP. This transformation can efficiently speed up the clustering convergence due to the following properties.

Theorem 15 *$F(\gamma)$ is convex.*

Proof: Suppose that $(\overline{\omega}_1, \cdots, \overline{\omega}_M)$ is an optimum of $F((1-\lambda)\gamma_1 + \lambda\gamma_2)$ with $\gamma_1 \neq \gamma_2$ and $0 \leqslant \lambda \leqslant 1$. $F((1-\lambda)\gamma_1 + \lambda\gamma_2) = f(\overline{\omega}_1, \cdots, \overline{\omega}_M) - ((1-\lambda)\gamma_1 + \lambda\gamma_2)g(\overline{\omega}_1, \cdots, \overline{\omega}_M) = \lambda(f(\overline{\omega}_1, \cdots, \overline{\omega}_M) - \gamma_2 g(\overline{\omega}_1, \cdots, \overline{\omega}_M)) + (1-\lambda)(f(\overline{\omega}_1, \cdots, \overline{\omega}_M) - \gamma_1 g(\overline{\omega}_1, \cdots, \overline{\omega}_M)) \leqslant \lambda \max_{\omega_1, \cdots, \omega_M} f(\omega_1, \cdots, \omega_M) - \gamma_2 g(\omega_1, \cdots, \omega_M) + (1-\lambda) \max_{\omega_1, \cdots, \omega_M} f(\omega_1, \cdots, \omega_M) - \gamma_1 g(\omega_1, \cdots, \omega_M) = \lambda F(\gamma_2) + (1-\lambda)F(\gamma_1)$. *Thus, $F(\gamma)$ is convex.*

Theorem 16 *$F(\gamma)$ is monotonically decreasing.*

Proof: Suppose that $\gamma_1 > \gamma_2$ and $(\overline{\omega}_1, \cdots, \overline{\omega}_M)$ is an optimal solution of $F(\gamma_1)$. Thus, $F(\gamma_1) = f(\overline{\omega}_1, \cdots, \overline{\omega}_M) - \gamma_1 g(\overline{\omega}_1, \cdots, \overline{\omega}_M) < f(\overline{\omega}_1, \cdots, \overline{\omega}_M) - \gamma_2 g(\overline{\omega}_1, \cdots, \overline{\omega}_M) \leqslant \max_{\omega_1, \cdots, \omega_M} f(\omega_1, \cdots, \omega_M) - \gamma_2 g(\omega_1, \cdots, \omega_M) = F(\gamma_2)$.

Theorem 17 *$F(\gamma) = 0$ has a unique solution.*

Proof: Based on the above-mentioned theorems, we know $F(\gamma)$ is continuous as well as decreasing. In addition, $lim_{\gamma \to +\infty} F(\gamma) = -\infty$ and $lim_{\gamma \to -\infty} F(\gamma) = +\infty$.

The procedure of solving this NPPP includes two parts: (1) find such a reasonable parameter γ ($F(\gamma) = 0$), making NPPP equivalent to NFPP; (2) given the parameter γ, solve a polynomial programming problem about the original variables $\omega_1, \cdots, \omega_M$. Our weight adjustment mechanism is an iterative procedure to find the solution of $F(\gamma) = 0$ and the corresponding weights after each clustering iteration. We first generate an initial matrix $\mathbf{P}^{(1)}$ with initial weights in terms of the scales of edge values in each vertex-centric path graph VG_m to produce an initial vertex clustering result through FCM [67] on the unified vertex-centric path graph VG. Based on the initial vertex clustering result, we construct an edge-centric path multigraph EMG_m for each edge-centric path graph EG_m. We then generate an initial edge clustering result on each EMG_m. According to the initial result of both vertex clustering and edge clusterings, we then calculate an initial $F(\gamma)$. Since $F(\gamma)$ is a monotonic decreasing function and $F(0) = \max_{\omega_1, \cdots, \omega_M} f(\omega_1, \cdots, \omega_M)$ is obviously non-negative, we start with an initial $\gamma = 0$ and solve the subproblem $F(0)$ by using existing fast polynomial programming model to update the weights $\omega_1, \cdots, \omega_M$. The parameter γ

Algorithm 3 Vertex/Edge-centric meta PATH graph Clustering

Input: M vertex-centric path graphs VG_m, M edge-centric path graphs EG_m, a clustering number K, and a parameter $\gamma^{(1)}$=0.
Output: vertex clustering membership matrix $\mathbf{X}$, M edge clustering membership matrices $\mathbf{Y}_1, \cdots, \mathbf{Y}_M$.

```
Initialize weights ω_1^(1), ..., ω_M^(1) in terms of the scales of edge values in each VG_m;
for t=1 to F(γ^(t)) converges to 0
  if t = 1
    Combine P_m of each VG_m into P^(t) of VG with Eq.(4.1);
    Invoke FCM to cluster vertices V_o in VG to generate X^(t) of VG;
  else
    Convert P_m of each VG_m into P_mk^(t) of each VMG_m with Eq.(4.4);
    Combine each VMG_m into VMG by computing all P_k^(t) in Eq.(4.13);
    Calculate S_k^(t) of VMG for each cluster c_k in Eqs.(4.14)-(4.15);
    Update X^(t) of VG with Eqs.(4.16)-(4.18);
  if t = 1
    Initialize Y_m^(t-1) of each EG_m with Eq.(4.6);
  Convert Q_m of each EG_m into Q_mk^(t) of each EMG_m with Eq.(4.5);
  Calculate T_mk^(t) of each EMG_m for each cluster c_k in Eqs.(4.7)-(4.8);
  Update Y_m^(t) of each EG_m with Eqs.(4.9)-(4.11);
  Compute O(X, Y_1, ..., Y_M, ω_1, ..., ω_M) in Eq.(4.19);
  Solve F(γ^(t)) in Eq.(4.22);
  Update ω_1^(t+1), ..., ω_M^(t+1);
  Refine γ^(t+1)=f(ω_1^(t+1), ..., ω_M^(t+1))/g(ω_1^(t+1), ..., ω_M^(t+1));
Return X^(t) and Y_1^(t), ..., Y_M^(t).
```

is gradually increased by $\gamma = f(\omega_1, \cdots, \omega_M)/g(\omega_1, \cdots, \omega_M)$ to help the algorithm enter the next round. The algorithm repeats the above-mentioned iterative procedure until $F(\gamma)$ converges to 0.

By assembling all the pieces in Section 4.4 together, we provide the pseudo code of our **VEPathCluster** algorithm in Algorithm 3.

4.5 Experimental Evaluation

We have performed extensive experiments to evaluate the performance of **VEPathCluster** on three real graph datasets.

4.5.1 Experimental Datasets

The first real dataset is extracted from the DBLP Bibliography data [1], which contains 112,483 authors (A), 728,497 papers (P), 2,633 venues (V), and 45,968 terms (T). We choose three meta paths: A-P-A, A-P-V-P-A and A-P-T-P-A, to cluster authors and three kinds of path edges into soft clusters simultaneously.

IMDb [2] is a searchable database of movies, TV and entertainment programs. We extract 48,975 actors (A), 31,188 movies (M), 4,774 directors (D), and 28 movie genres (G) from the original IMDb dataset. Three candidate meta paths: A-M-A, A-M-D-M-A and A-M-G-M-A, are used to assign each actor and three types of path edges to soft clusters.

The third real-world dataset is extracted from the Yelp's academic dataset [3], which includes 15,715 businesses (B), 470,212 reviews (R), 138,969 users (U), and 30,475 review terms (T). We select two meta paths: B-R-U-R-B and B-R-T-R-B, to generate the soft clusterings of businesses and two kinds of path edges.

4.5.2 Comparison Methods and Measures

We compare **VEPathCluster** with two representative soft clustering algorithms, **Fuzzy C-Means** (FCM) [67], **Gustafson-Kessel** (GK) [70], and one recently developed method **PathSelClus** [22]. For the first two clustering methods, we add the adjacency matrices of all vertex-centric path graphs together to get one single matrix. The first two methods perform vertex-centric soft clustering on a single graph and PathSelClus performs vertex-centric soft clustering on multiple graphs based on the assumption of vertex homophily.

We also evaluate three partial versions of VEPathCluster to show the strengths of edge clustering and weight learning respectively: (1)**VEPathCluster-VE** with only vertex clustering and edge clustering; (2) **VEPathCluster-VW** with only vertex clustering and weight update; and (3) **VEPathCluster-EW** with only edge clustering and weight update.

[1] http://dblp.uni-trier.de/xml/
[2] http://www.imdb.com/interfaces
[3] http://www.yelp.com/academic_dataset

Evaluation Metrics We use three measures to evaluate the quality of vertex clustering by different methods. The fuzzy Dunn index [71, 72] is defined as the ratio between the minimal fuzzy intra-cluster similarity and the maximal fuzzy inter-cluster similarity.

$$Dunn(\mathbf{X}) = \frac{\min\limits_{1\le k\le K}\left(\frac{1}{(\sum_{i=1}^{N_{V_c}}\mathbf{X}_k(v_i))(\sum_{j=i+1}^{N_{V_c}}\mathbf{X}_k(v_j))}\sum_{i=1}^{N_{V_c}}\sum_{j=i+1}^{N_{V_c}}\mathbf{X}_k(v_i))\mathbf{X}_k(v_j))\mathbf{P}(v_i,v_j)\right)}{\max\limits_{1\le k<l\le K}\left(\frac{1}{(\sum_{i=1}^{N_{V_c}}\mathbf{X}_k(v_i))(\sum_{j=1}^{N_{V_c}}\mathbf{X}_l(v_j))}\sum_{i=1}^{N_{V_c}}\sum_{j=1}^{N_{V_c}}\mathbf{X}_k(v_i))\mathbf{X}_l(v_j))\mathbf{P}(v_i,v_j)\right)} \tag{4.23}$$

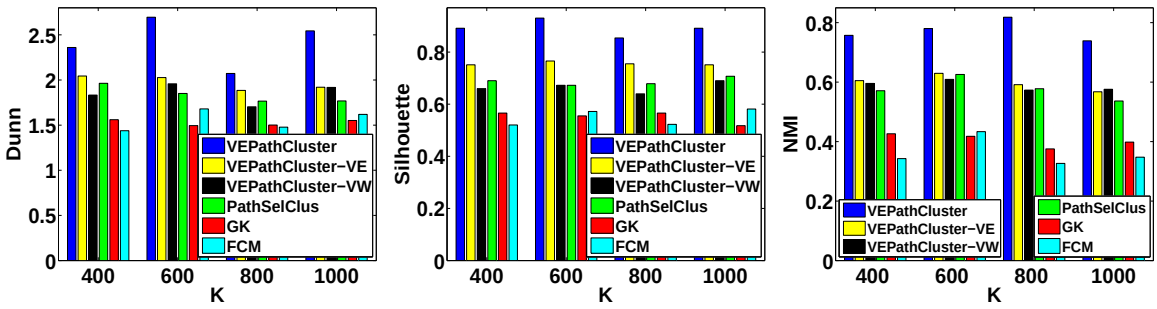
where $\mathbf{X}$ is the vertex soft clustering membership matrix and $Dunn(\mathbf{X})$ is bounded in the range $[0, +\infty)$. A larger value of $Dunn(\mathbf{X})$ indicates a better clustering.

The following two metrics are often used to evaluate the hard clustering result, we thus map the soft clustering results by various methods into hard clustering results with the maximum probability of each vertex as its hard cluster labels.

$$\begin{aligned} b(v_i) &= \frac{1}{|VC_k|-1}\sum_{v_j\in VC_k, j\ne i}\mathbf{P}(v_i,v_j), \quad a(v_i) = \max_{1\le l\le K, l\ne k}\left(\frac{1}{|VC_l|}\sum_{v_j\in VC_l}\mathbf{P}(v_i,v_j)\right) \\ Silhouette(\{VC_k\}_{k=1}^K) &= \frac{1}{K}\sum_{k=1}^{K}\left(\frac{1}{|VC_k|}\sum_{v_i\in VC_k}\frac{b(v_i)-a(v_i)}{\max\{a(v_i),b(v_i)\}}\right) \end{aligned} \tag{4.24}$$

where $\{VC_k\}_{k=1}^K$ represents the mapped hard clustering of the target vertices V_c, i.e., $V_c = \bigcup_{k=1}^K VC_k$ and $VC_k \bigcap VC_l = \phi$ for $\forall 1 \le k, l \le K, k \ne l$. $\mathbf{P}(v_i, v_j)$ is the edge value between two vertices v_i and v_j in the unified vertex-centric path graph VG. The silhouette coefficient [73] with the bound of [-1, 1] contrasts the average intra-cluster similarity with the average inter-cluster similarity. The larger the value, the better the quality.

Following the same strategy used in [22], we use $NMI(X, Y) = \frac{I(X;Y)}{\sqrt{H(X)H(Y)}}$ to compare the generated vertex clustering with the ground truth, where X and Y represent two cluster label vectors for the ground truth clustering and the calculated clustering by a clustering method respectively. $NMI(X, Y)$ is in the interval $[0, 1]$ and a larger NMI value indicates a



(a) Dunn (b) Silhouette (c) NMI

Figure 4.7: Vertex Clustering Quality on DBLP

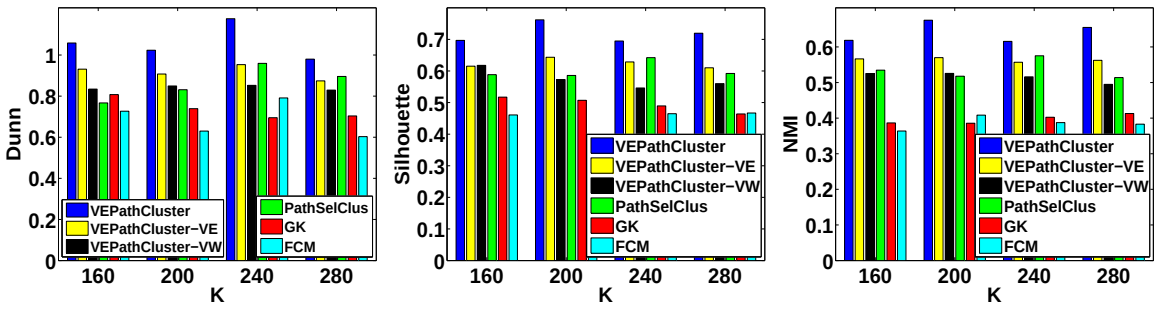
better clustering.

Similarly, we use the same three measures to evaluate the quality of edge clustering by VEPathCluster. We report the average metric value for each measure based on M edge clustering results.

4.5.3 Vertex Clustering Quality

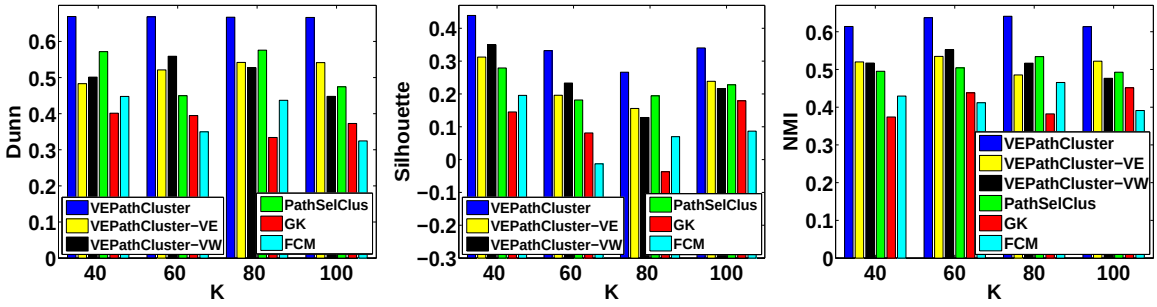
Figures 4.7-4.9 exhibit the vertex clustering quality on DBLP, IMDb and Yelp by varying the number of clusters. We divide six soft clustering methods into three categories: (1) FCM and GK perform the basic vertex clustering only based on the matrix of the unified vertex-centric path graph; (2) PathSelClus, VEPathCluster-VW and VEPathCluster-VE utilize partial optimization techniques to further improve the quality of vertex clustering; and (3) VEPathCluster makes use of both techniques of edge clustering and weight learning to achieve the promotion as much as possible.

First, PathSelClus, VEPathCluster-VW and VEPathCluster-VE significantly outperform FCM and GK on all three evaluation measures. We know that the edges in different vertex-centric path graphs usually have values with different scales. As vertex-centric clustering methods, both PathSelClus and VEPathCluster-VW efficiency integrates the matrices of multiple vertex-centric path graphs through the iterative weight learning mechanism to learn the optimal weight assignment for these matrices. Thus, the measure scores obtained by them are often comparable to each other. On the other hand, VEPathCluster-VE inte-



(a) Dunn (b) Silhouette (c) NMI

Figure 4.8: Vertex Clustering Quality on IMDb



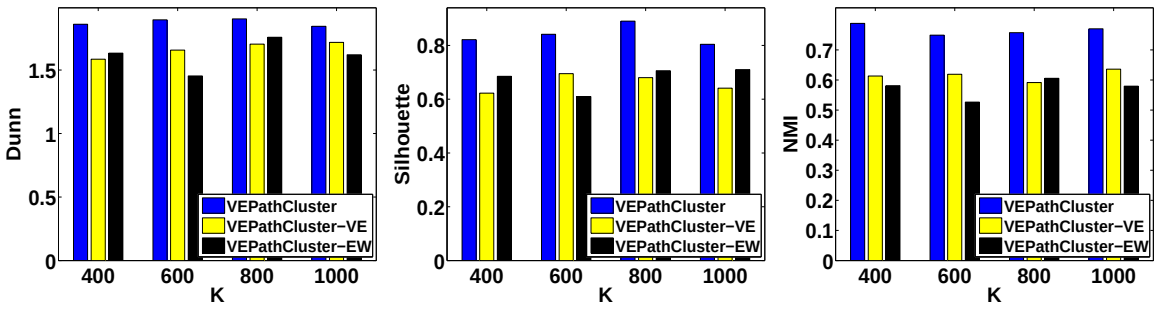
(a) Dunn (b) Silhouette (c) NMI

Figure 4.9: Vertex Clustering Quality on Yelp

grates vertex clustering and edge clustering to mutually enhance each other. These results demonstrate that the importance of exploiting both edge clustering and weight learning for meta path graph clustering.

Second, it is observed that VEPathCluster-VE outperforms PathSelClus and VEPathCluster-VW on three graph datasets, even though the dynamic weight refinement is not used in VEPathCluster-VE while both PathSelClus and VEPathCluster-VW employed some iterative weight learning method to find the optimal weight assignment and improve the clustering quality. This is because both PathSelClus and VEPathCluster-VW are based solely on vertex homophily, without incorporating and integrating edge homophily into the clustering analysis. These results illustrate that employing edge clustering is more important than exploit weight learning in solve the meta path graph clustering problem.

Finally, among all six clustering methods, VEPathCluster achieves the best clustering



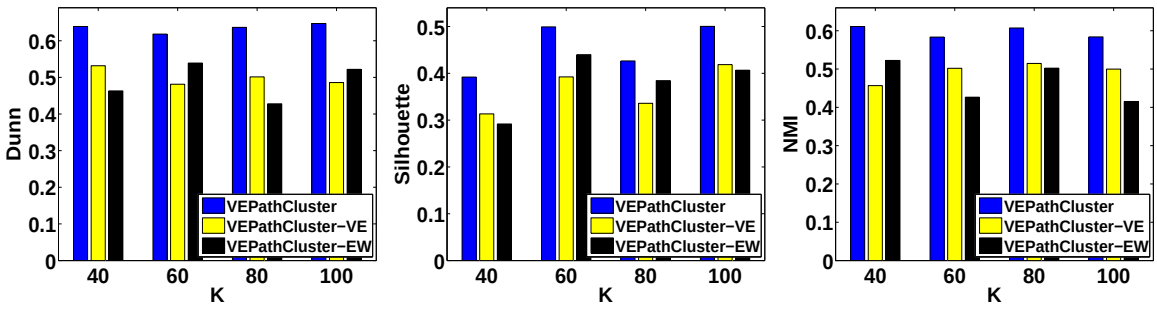
(a) Dunn (b) Silhouette (c) NMI

Figure 4.10: Edge Clustering Quality on DBLP

performance for all three evaluation measures in most cases. Compared to other algorithms, VEPathCluster averagely achieves 18.7% Dunn increase, 14.1% Silhouette boost and 22.4% NMI improvement on DBLP, 10.6% Dunn growth, 10.4% Silhouette increase and 8.7% NMI boost on IMDb, and 17.7% Dunn increase, 23.9% Silhouette boost and 11.6% NMI improvement on Yelp, respectively. Concretely, there are three critical reasons for high accuracy of VEPathCluster: (1) the clustering-based multigraph model integrates both vertex-centric clustering and edge-centric clustering to accurately capture the cluster-specific relationships between vertices and between edges; (2) the edge-centric random walk model provides a natural way to capture the dependencies among path edges within each vertex-centric path graph; and (3) the iterative learning algorithm help the clustering model achieve a good balance among different types of vertex-centric path graphs and edge-centric path graphs.

4.5.4 Edge Clustering Quality

Given that FCM, GK, PathSelClus and VEPathCluster-VW are vertex-centric soft clustering methods, we skip the experimental evaluation of edge clustering for these four approaches. Figures 4.10-4.11 present the edge clustering quality by three versions of VEPathCluster on two datasets with different K respectively. Similar trends are observed for the edge clustering quality comparison: VEPathCluster achieves the largest Dunn values (>0.62), the highest Silhouette around 0.39-0.89, and the largest NMI (>0.58), which



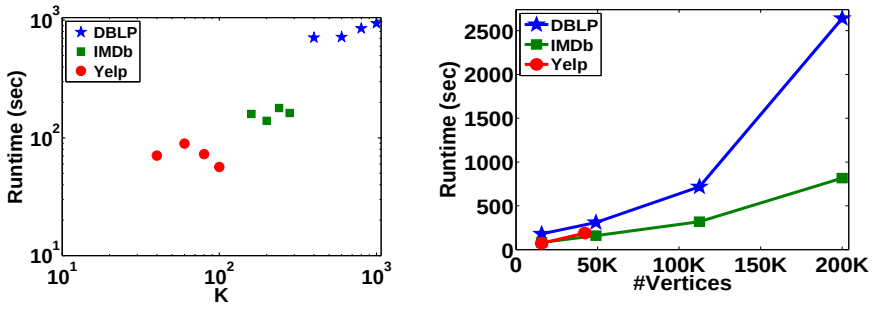
(a) Dunn (b) Silhouette (c) NMI

Figure 4.11: Edge Clustering Quality on Yelp

are obviously better than other two methods. As K increases, the measure scores achieved by VEPathCluster remains relatively stable, while the measure scores of other two methods oscillate in a fairly large range. In addition, in terms of three evaluation measures, VEPathCluster-VE outperforms VEPathCluster-EW in some cases but VEPathCluster-EW performs better than VEPathCluster-VE in some cases. These results demonstrate that each of vertex clustering, edge clustering and weight learning plays an important role in meta path clustering. Thus, we should integrate three optimization techniques to further improve the clustering quality.

4.5.5 Clustering Efficiency

Figure 6.5 (a) presents the clustering time achieved by VEPathCluster on DBLP, Last.fm and IMDb with the same K setups in the experiments of clustering quality in Figures 4.7-4.11 respectively. Figure 6.5 (b) exhibits the scalability test of VEPathCluster by varying the number of target vertices on three datasets respectively. For DBLP and IMDb, we test four different setups of #Vertices, i.e., #Vertices = 15,715, 48,975, 112,483, 200,000 respectively. However, we only test #Vertices = 15,715, 42,153 for Yelp since the original Yelp dataset contains up to 42,153 businesses. We observe that VEPathCluster scales well with the size of graph for different graph datsets and shows good performance with varying K. A careful examination reveals that the bottleneck component of the overall time complexity for VEPathCluster is the execution time of iterative vertex clustering and edge



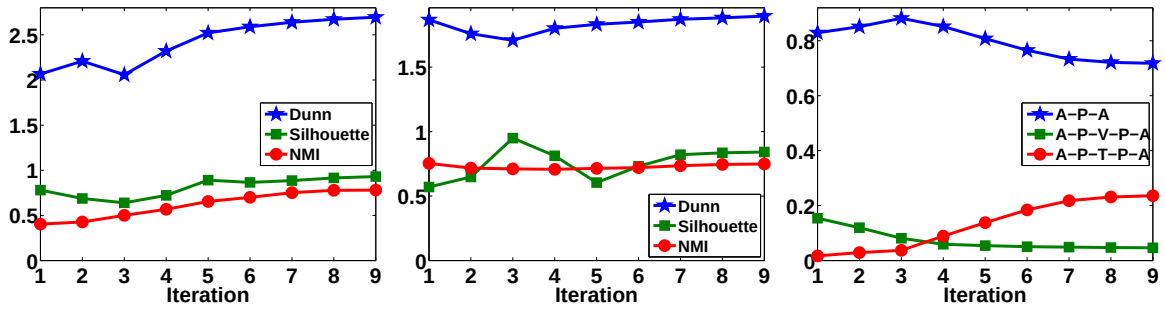
(a) Varying K (b) Varying #Vertices

Figure 4.12: Clustering Efficiency

clusterings, which mainly consist of a series of matrix-vector multiplications. Let K be the number of clusters, N_{V_c} be the number of target vertices in the unified vertex-centric path multigraph, N_{E_k} $(1 \leq k \leq K)$ be the number of parallel edges on the k^{th} cluster in the unified vertex-centric path multigraph, M be the number of edge-centric path multigraphs, N_{E_m} $(1 \leq m \leq M)$ be the number of vertices in the m^{th} edge-centric path multigraph, $N_{F_{mk}}$ $(1 \leq k \leq K)$ be the number of parallel edges on the k^{th} cluster in the m^{th} edge-centric path multigraph, t_i is the number of inner iterations, and t_o be the number of outer iterations in the clustering process. At the worst case, i.e., the original graph dataset is relatively dense, the complexity of performing vertex clustering on the unified vertex-centric path multigraph is equal to $O(t_o t_i K N_{V_c}^2)$ and the cost of performing edge clustering on each of M edge-centric path multigraphs is equal to $O(t_o \sum_{m=1}^{M} t_i K N_{E_m}^2)$. However, when the original graph dataset is very spare, the complexity of matrix-vector multiplication is approximately bounded by the size of edges. In this situation, the complexity of performing vertex clustering is reduced to $O(t_o t_i \sum_{k=1}^{K} N_{E_k})$ and the cost of performing edge clustering on all M edge-centric path multigraphs is decreased to $O(t_o \sum_{m=1}^{M} t_i \sum_{k=1}^{K} N_{F_{mk}})$.

4.5.6 Clustering Convergence

Figure 4.13 (a) and (b) exhibit the convergence trend of vertex clustering and edge clustering in terms of three evaluation measures on DBLP. Both the Dunn values and the NMI scores in two figures keep increasing or relatively stable and have convex curves when we



(a) Vertex Clustering (b) Edge Clustering (c) Weight Assignment

Figure 4.13: Clustering Convergence

iteratively perform the tasks of vertex clustering, edge clustering and weight update during the clustering process. On the other hand, the Silhouette values first fluctuate slightly within a range of [0.57, 0.95] and then converge very quickly. The entire clustering process converges in nine iterations for DBLP. Figure 4.13 (c) shows the tendency of weight update for three meta paths on DBLP. We keep the constraint of weights for three meta paths unchanged, i.e., $\sum_{m=1}^{M} \omega_m = 1$, during the clustering process. We observe that all three weights converge as the clustering process converges. An interesting phenomenon is that the weight for the A-P-A meta path first increases and then decreases with the iterations, the weight for the A-P-V-P-A meta path keeps decreasing and the weight curve for the A-P-T-P-A meta path has a converse trend. A reasonable explanation is that people who have many publications on the same conferences may have different research topics but people who have many papers with the same terms usually have the same research interests. On the other hand, for a pair of coauthors, their primary research areas are not always consistent in terms of the number of their coauthored papers, as illuminated in the example in Figure 4.2. Another interesting finding is that the weight for the A-P-A meta path is relatively large and other two weights are fairly small. This is because that the edges in different path graphs usually have values with different scales, as shown in Figure 4.3. In addition, the length of either of other two meta paths is larger than that of the A-P-A meta path, and there are many venues and terms in the DBLP dataset. To maintain a good balance among different meta paths, the algorithm needs to set larger weights for the path graphs with small-scale

Table 4.1: Cluster Membership Probabilities of Authors Based on Three Meta Paths from DBLP

Author/Cluster	DB	DM	AI	IR
Ming-Syan Chen	0.258	0.588	0.021	0.134
W. Bruce Croft	0.058	0.006	0.026	0.909
Christos Faloutsos	0.346	0.539	0.012	0.102
Jiawei Han	0.373	0.459	0.057	0.111
H. V. Jagadish	0.904	0.048	0.014	0.034
Laks V. S. Lakshmanan	0.809	0.128	0.011	0.053
Hector Garcia-Molina	0.810	0.028	0.021	0.141
Eric P. Xing	0.009	0.123	0.830	0.038
Qiang Yang	0.012	0.265	0.512	0.210
Philip S. Yu	0.358	0.507	0.027	0.108
Chengqi Zhang	0.023	0.744	0.140	0.093

edges to maintain their contributions to clustering.

4.5.7 Case Study

We examine some details of the experiment results based on DBLP. Table 4.1 exhibits the set of authors and their cluster membership probabilities after nine iterations based on three meta paths: A-P-A, A-P-V-P-A and A-P-T-P-A. We only present most prolific DBLP experts in the area of database (DB), data mining (DM), artificial intelligence (AI) and information retrieval (IR). We observe that the predicted cluster memberships of authors are consistent with their actual research areas. For those researchers known to work in multiple research areas, the cluster membership distributions also correspond to their current research activities. For example, both Jiawei Han and Philip S. Yu are experts on data mining and database, though their DM probabilities are slightly higher since each of them and their circle of co-authors have more DM papers. Table 4.2 shows the set of path edges between the above authors in the A-P-A vertex-centric path graph and their cluster membership probabilities after nine clustering iterations. We have observed that most of author pairs associated to path edges usually have different primary research areas, e.g., the primary research areas of W. Bruce Croft and Hector Garcia-Molina are IR and DB respectively. In this situation, the cluster favorite of the path edges between the pairwise authors are

Table 4.2: Cluster Membership Probabilities of A-P-A Path Edges from DBLP

Path Edge/Cluster	DB	DM	AI	IR
(Ming-Syan Chen, Philip S. Yu)	0.630	0.284	0.023	0.063
(W. Bruce Croft, Hector Garcia-Molina)	0.702	0.035	0.065	0.199
(Christos Faloutsos, H. V. Jagadish)	0.547	0.365	0.017	0.072
(Christos Faloutsos, Eric P. Xing)	0.238	0.713	0.015	0.034
(Jiawei Han, Laks V. S. Lakshmanan)	0.624	0.356	0.006	0.013
(Jiawei Han, Philip S. Yu)	0.518	0.424	0.013	0.045
(Qiang Yang, Philip S. Yu)	0.083	0.785	0.131	0.001
(Qiang Yang, Chengqi Zhang)	0.023	0.684	0.228	0.065

often dominated by the primary research area of one associated author. For example, the path edge (W. Bruce Croft, Hector Garcia-Molina) has a main cluster favorite of DB. An interesting phenomenon is that although both Ming-Syan Chen and Philip S. Yu are experts on data mining, i.e., they both have more research publications in the area of data mining than in any other academic area such as database. However, the path edge (Ming-Syan Chen, Philip S. Yu) have a large probability on cluster DB. A careful examination reveals that most of coauthored publications between two experts are database specific.

4.6 Conclusions

We have presented a meta path graph clustering framework for mining heterogeneous information networks. First, we model a heterogeneous information network containing multiple types of meta paths as multiple vertex-centric path graphs and multiple edge-centric path graphs. Second, we cluster both vertex-centric path graph and edge-centric path graphs to generate vertex clustering and edge clusterings. Third, a reinforcement algorithm is provided to tightly integrate vertex clustering and edge clustering by mutually enhancing each other. Finally, an iterative learning strategy is proposed to dynamically refine both clustering results by continuously learning the degree of contributions of different path graphs.

CHAPTER 5

GRAPHTWIST: FAST ITERATIVE GRAPH COMPUTATION WITH COMPUTATION-TIER OPTIMIZATION

Large-scale real-world graphs are known to have highly skewed vertex degree distribution and highly skewed edge weight distribution. Existing vertex-centric iterative graph computation models suffer from a number of serious problems: (1) poor performance of parallel execution due to inherent workload imbalance at vertex level; (2) inefficient CPU resource utilization due to short execution time for low-degree vertices compared to the cost of in-memory or on-disk vertex access; and (3) incapability of pruning insignificant vertices or edges to improve the computational performance. To address the above technical challenges, this chapter presents a scalable, efficient, and provably correct two-tier graph parallel processing framework, GraphTwist. At storage and access tier, GraphTwist maximizes parallel efficiency by employing three graph parallel abstractions for partitioning a big graph by slice, strip or dice based partitioning techniques. At computation tier, GraphTwist presents two utility-aware pruning strategies: slice pruning and cut pruning, to further improve the computational performance while preserving the computational utility defined by graph applications. Theoretic analysis is provided to quantitatively prove that iterative graph computations powered by utility-aware pruning techniques can achieve a very good approximation with bounds on the introduced error. Our experimental results also show that GraphTwist can deliver significant speedup, while maintaining high quality, for big graph analysis compared to existing representative approaches.

5.1 Introduction

Graph as an expressive data structure is popularly used to model structural relationship between objects in many application domains, such as social networks, web graphs, RD-

F graphs, sensor networks, protein interaction networks. These graphs typically consist of millions of vertices and billions of edges. Efficient iterative computation on such huge graphs is widely recognized as a challenging big data research problem, which has received heated attention recently. We can broadly classify existing research activities on scaling iterative graph computations into two categories: (1) Distributed solutions and (2) Single PC based solutions. Most of existing research efforts are dedicated to the distributed graph partitioning strategies that can effectively break large graphs into small, relatively independent parts [74, 75, 76, 77, 78, 79]. Several recent efforts [80, 81, 82, 83, 84] have successfully demonstrated huge opportunities for optimizing graph processing on a single PC through graph parallel abstractions that are efficient in both storage organization and in-memory computation.

However, to the best of our knowledge, existing approaches fail to address the following challenges:

- **Efficient vertex-oriented aggregation.** Existing graph abstractions mainly focus on parallelism strategies at each vertex and its associated edges (its adjacency list). However, the vertex-centric parallel tasks dramatically increase the inter-vertex communication overhead regardless whether the parallel tasks are executed in local memory or shared memory across a cluster of compute nodes. Thus, how to perform vertex-oriented aggregation operations at higher graph parallel abstractions to improve the efficiency remains to be very challenging for big graphs.
- **Efficient handling of large graphs with highly skewed vertex degree distribution and highly skewed edge weight distribution.** Large-scale graphs typically have skewed vertex degree distribution. Concretely, a relatively small number of vertices connect to a large fraction of graph, but a large number of vertices have very few or no links to other vertices. Some real-world graphs, say DBLP coauthor graph, have skewed edge weight distribution. Partitioning a large graph in terms of vertex partitions without considering skewed vertex degree distribution or edges with skewed weight distribution may result in

substantial workload imbalance in parallel computation. We argue that different graph parallel abstractions should be supported at both access tier and computation tier to promote well-balanced computation workloads, better CPU resource utilization, and more efficient graph parallel executions.

- **Resource-adaptive partitioning of big graphs.** Given that different types of iterative graph applications combined with different sizes of graphs often have different resource demands on CPU, memory and disk I/O, choosing the right granularity of graph parallel abstractions allows big graph analysis to be performed on any commodity PC. An important technical challenge is to devise tunable multi-level graph parallel abstractions such that the graph processing system enables more balanced parallel workloads, more effective resource utilization for parallel computation, and at the same time offer the best access locality by maximizing sequential access and minimizing random access.
- **Exploring graph utility-aware pruning techniques.** Iterative graph computations on large graphs with millions of vertices and billions of edges can generate the intermediate results that are orders of magnitude bigger than the original graph. One way to improve the computational performance on such big graphs is to prune those vertices or edges that do not directly contribute to the utility of graph computation as early as possible. For example, when computing the social influence of authors in the area of DB on the DBLP coauthor graph, those coauthor edges that are not DB specific can be pruned at early stage. Similarly, when computing the PageRank scores, those smaller elements in the transition matrix can be pruned in the subsequent iterations. Such pruning can significantly speed up the iterative graph computations while maintaining the desired graph utility. An important challenge is to design a graph storage data structure to flexibly support such utility-aware progressive pruning techniques.

To address these new challenges, we develop GraphTwist, an innovative graph parallel abstraction model with two-tier optimizations for processing big graphs with skewed vertex degree distribution and skewed edge weight distribution. The main contributions of this

chapter are summarized below.

- We present a scalable two-tier graph parallel aggregation framework, GraphTwist, for efficiently executing complex iterative computation tasks over large-scale graphs on a shared-memory multiprocessor computer.
- At storage and access tier, we propose a compact data structure for big graphs augmented with three divide-and-conquer graph parallel abstractions. We introduce the index structure, the basic algebra and its core operations to support resource-adaptive graph partitioning by selecting the right granularity of parallel abstractions based on both the system capacity and the utility of graph algorithm.
- At computation tier, we present two utility-aware pruning strategies: slice pruning and cut pruning, to further improve the performance by removing non-contributing vertices or edges while preserving the computational utility with bounds on the introduced error.
- Empirical evaluation over real large graphs demonstrates that GraphTwist outperforms existing representative graph parallel models in terms of both effectiveness and efficiency.

5.2 Related Work

We classify existing research activities on graph processing system into two broad categories below [85, 74, 80, 86, 87, 88, 89, 75, 90, 76, 81, 77, 78, 82, 84, 83, 91, 79, 92, 93, 94].

Single PC based systems [80, 81, 82, 84, 83, 92, 93] are gaining attention in recent years. GraphLab [80] presented a new sequential shared memory abstraction where each vertex can read and write data on adjacent vertices and edges. It supports the representation of structured data dependencies and flexible scheduling for iterative computation. GraphChi [81] partitions a graph into multiple shards by storing each vertex and its in-edges in one shard. It introduces a novel parallel sliding window based method to facilitate fast access to the out-edges of a vertex stored in other shards. TurboGraph [82] presented a multi-thread graph engine by using a compact storage of slotted page list and exploiting the

full parallelism of multi-core CPU and Flash SSD I/O. X-Stream [83] is an edge-centric approach to the scatter-gather programming model on a single shared-memory machine. It uses streaming partitions to utilize the sequential streaming bandwidth of the storage medium for graph processing.

Distributed graph systems [85, 74, 87, 75, 76, 78, 91, 79, 94] have attracted active research in recent years, with Pregel [74], PowerGraph [76]/Distributed GraphLab [75], and GraphX [79] as the most popular systems. Pregel [74] is a bulk synchronous message passing abstraction where vertices can receive messages sent in the previous iteration, send messages to other vertices and modify its own state and that of its outgoing edges or mutate graph topology. PowerGraph [76] extends GraphLab [80] and distributed GraphLab [75] by using the Gather-Apply-Scatter model of computation to address the natural graphs with highly skewed power-law degree distributions. GraphX [79] enables iterative graph computation, written in Scala like API in terms of GraphX RDG, to run on the SPARK [95] cluster platform, making the programming of iterative graph algorithms on Spark easier than PowerGraph and Pregel.

Iterative graph applications has been extensively studied in the areas of machine learning, data mining and information retrieval [96, 97, 28, 98, 65, 99, 100, 3, 101, 43, 14, 20, 51, 102, 103, 104, 105]. Typical examples of real-world iterative graph applications include ranking, similarity search, graph classification, graph clustering, and collaborative filtering. Popular iterative graph applications can be categorized into three classes in terms of the core computation used in the respective algorithms: (1) matrix-vector computation, such as PageRank [96], EigenTrust [99] and Random Walk with Restart [100]; (2) matrix-matrix computation, including Heat Diffusion Kernel [97, 3], Label Propagation [101], wvRN [43], Markov Clustering [14] and SA-Cluster [20]; and (3) matrix factorization, such as NMF [102], SVD++ [103], Social Regularization [104]. They often need to repeatedly self-interact on a single graph or iteratively interact among multiple graphs to discover both direct and indirect relationships between vertices.

Table 5.1: Real-world Datasets

Graph	Type	#Vertices	#Edges	AvgDeg	MaxIn	MaxOut
Yahoo [106]	simple graph	1.4B	6.6B	4.7	7.6M	2.5K
Twitter [107]	simple graph	41.7M	1.5B	35.25	770.1K	3.0M
Facebook [108]	simple graph	5.2M	47.2M	18.04	1.1K	1.1K
DBLPS [109]	simple graph	1.3M	32.0M	40.67	1.7K	1.7K
DBLPM [109]	multigraph	0.96M	10.1M	21.12	1.0K	1.0K
Last.fm [110]	multigraph	2.5M	42.8M	34.23	33.2K	33.2K

To our best knowledge, GraphTwist is the first one to support multi-level programmable parallel graph abstractions (slice, strip, dice) and to provide resource adaptive selection of the right level of graph parallel granularity for partitioning, storing and accessing large graphs.

5.3 Access-tier Optimization

Large-scale graphs often have skewed vertex degree distribution. Table 5.1 shows the vertex degree distributions of several real graph datasets used in our experimental evaluation. For example, the Yahoo dataset has average vertex degree of 4.7 but maximum indegree of 7.6 million and maximum outdegree of 2.5 thousand. Similar observations can be made on several other datasets. Clearly, by replying on the simple vertex block based graph partitioning (i.e., each vertex and its adjacency list is in one partition), existing graph parallel models may result in very poor parallel performance due to substantial workload imbalance at vertex level in parallel computation. In addition, the processing time for vertices with small degree is very short compared to the in-memory and on-disk access latency, leading to inefficient CPU utilization in addition to poor workload balance.

Although many graph datsets (Yahoo, Twitter and Facebook) originally have 0/1 edge weights or do not have explicit edge weights, some real graphs (DBLP) have skewed edge weight distribution. In addition, in many iterative graph applications, we need to transform the original graphs into the weighted graphs. A typical example is the transition matrix of graph, which is widely applied to many real applications, such as PageRank, graph clustering and graph classification. Consider PageRank as an example, we need to iteratively

calculate the multiplication between the transition matrix and the ranking vector. However, the transition matrix often has skewed edge weight distribution and small weight values may contribute little to the overall quality of the PageRank ranking result. Thus, GraphTwist is designed to address these issues by introducing two-tier optimizations to speed up iterative graph computations. The low-tier optimization utilizes a compact and flexible data structure and enables access locality optimization (Section 5.3). We introduce a graph utility-aware filtering method as the high-tier optimization, while preserving the desired quality with provable error bound (see Section 5.4 for detail). Our two-tier optimization model is designed by enabling careful interaction between access tier and computation tier for fast iterative graph computation. The storage and access-tier optimizations in GraphTwist enable us to exercise the optimizations at computation tier more effectively.

5.3.1 Graph Processing with 3D Cube

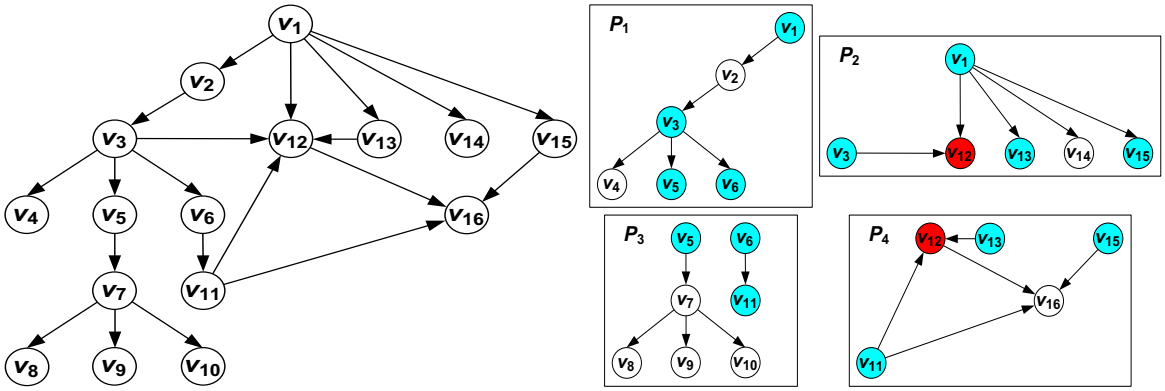
3D Cube. GraphTwist represents a given graph G as a 3D cube I with source vertices, destination vertices and edge weights as the three dimensions. Formally, a directed graph is defined as $G=(V, E, W)$ where V is a set of n vertices, E is a set of directed edges, and W is a set of weights of edges in E. Each vertex is associated with one or more states. Two vertices may be connected by multiple parallel edges. For an edge $e=(u, v)\in E$, we refer to e as the in-edge of v and the out-edge of u and we refer to u and v as the source vertex and the destination vertex of e respectively. In GraphTwist, we model a graph G with a 3-dimensional representation of G, called **3D cube**, denoted as $I=(S, D, E, W)$ where $S=V$ represents the set of source vertices and $D=V$ specifies the set of destination vertices. Given a vertex $u\in S$ and a vertex $v\in D$, if $(u, v)\in E$ then $(u, v).weight=w\in W$ and (u, v, w) represents a cell with u, v, w as coordinates.

GraphTwist by design provides three alternative graph parallel abstractions to partition a graph and to enable locality-optimized access to the stored graph using three different levels of granularity: slice, strip and dice. By utilizing different graph parallel abstractions

based on memory resource capacity, GraphTwist can process big graphs efficiently on a single PC with different resource capacity, by using a unified multi-level graph parallel abstractions based computation framework.

Dice-based Parallel Graph Abstraction

The dice partitioning method partitions a large graph G into dice-based subgraph blocks and store G in dices to balance the parallel computation tasks and maximize the advantage of parallel processing. Concretely, given G=(V, E, W) and its 3D cube I=(S, D, E, W), we first sort the vertices in S and D by the lexical order of their vertex IDs. Then we partition the destination vertices D into q disjoint partitions, called **destination-vertex partitions** (DVPs). Similarly, we partition the source vertices S ($|D|$=$|V|$) into r disjoint partitions, called **source-vertex partitions** (SVPs). A **dice** of I is a subgraph of G, denoted as H=(S_H, D_H, E_H, W_H), satisfying the following conditions: $S_H \subseteq S$ is one of the SVP, denoting a subset of source vertices, $D_H \subseteq D$ is one of the DVP, denoting a subset of destination vertices, $W_H \subseteq W$ is a subset of edge weights, and E_H=$\{(u, v)|u \in S_H, v \in D_H, (u, v) \in E, (u, v).$ $weight \in W_H\}$ is a set of directed edges, each with its source vertex from S_H and its destination vertex from D_H and its edge weight in W_H. Unlike a vertex and its adjacency list (edges), a dice is a subgraph block comprised of a SVP, a DVP and the set of edges that connect source vertices in the SVP to the destination vertices in the DVP. Thus, a high degree vertex u and its edges are typically partitioned into multiple dices. Figure 5.1 (a) gives an example graph and Figure 5.1 (b) shows a dice-based partitioning of this example graph. Each of four dice partitions is a dice-based subgraph satisfying the constraint defined by the specific SVP and DVP. Because in-edges and out-edges of a vertex are often applied to different application scenarios, we maintain two types of dices for each vertex v in G: one is **in-edge dice** (IED) containing only in-edges of v and another is **out-edge dice** (OED) containing only out-edges of v. Figure 5.2 shows the storage organization of the dice partitions in Figure 5.1 (b), consisting of a vertex table, an edge table (in-edges or



(a) Original Graph (b) Dice Partitioning

Figure 5.1: Dice Partitioning: An Example

out-edges), and a mapping of vertex ID to partition ID.

In GraphTwist, dice is the smallest storage unit and by default the original graph G is stored on disk in unordered dices. To provide efficient processing for all types of iterative graph computations, GraphTwist stores an original graph using two types of 3D cubes: in-edge cube and out-edge cube, each consists of unordered set of dices of the same type on disk (IEDs or OEDs). This provides efficient access locality for iterative graph computations that require only out-edges or only in-edges or both.

Slice-based Parallel Graph Abstraction

In contrast to dices, slices are the largest partition units in GraphTwist. To address the skewed edge weight distribution, we provide the slice-based graph partitioning method, which partitions a 3D cube of graph into p slices along dimension W. p is chosen such that edges with similar weights are clustered into the same partition. Formally, given a 3D cube I=(S, D, E, W), a **slice** of I is denoted as J=(S, D, E_J, W_J) where $W_J \subseteq W$ is a subset of edge weights, and E_J=$\{(u, v)|u \in S, v \in D, (u, v).weight \in W_J, (u, v) \in E\}$ is a set of directed edges from S to D with weights W_J. A big advantage of slice partitioning along dimension W is that we can choose those slices that meet the utility requirement to carry out the iterative graph computation according to application-dependent accuracy requirements.

An intuitive example for utilizing slice partitioning is to handle multigraphs. A multi-

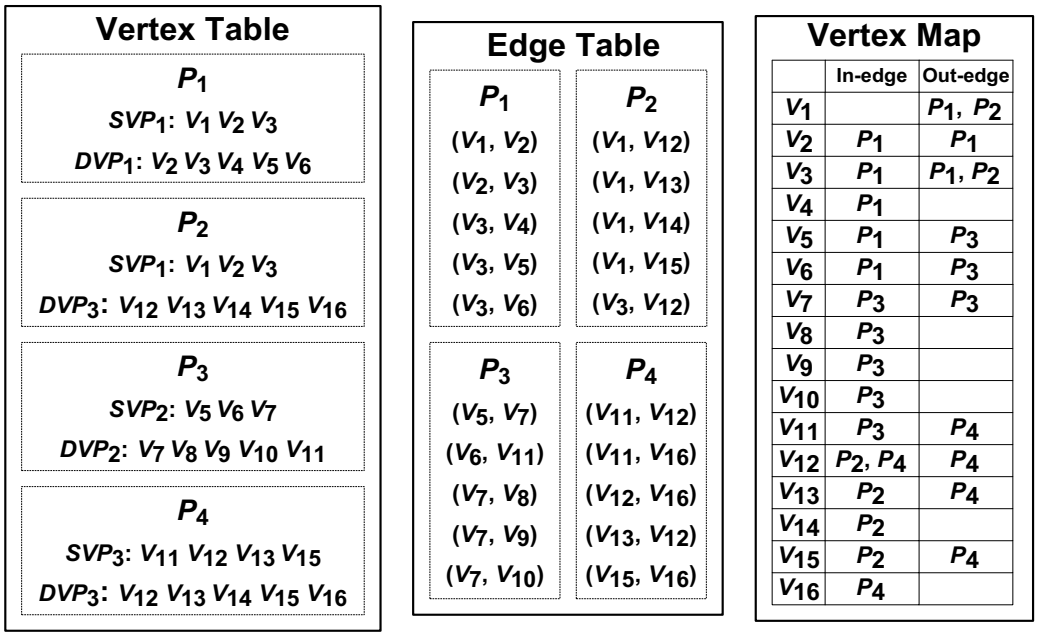
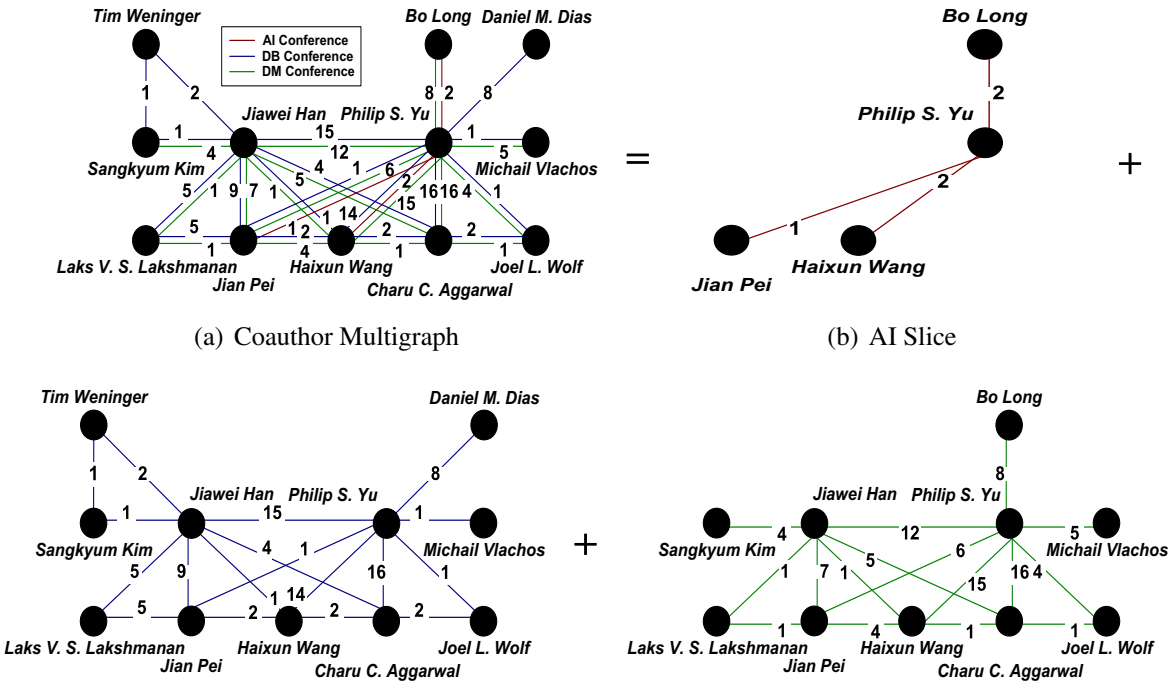


Figure 5.2: Dice Partition Storage (OEDs)

graph is a graph that allows for parallel edges (multiple edges) between a pair of vertices. RDF graph is a typical example of multigraph, where a pair of subject and object vertices may have multiple edges with each annotated by one predicate. Similarly, the DBLP coauthor graph can also be generated as a coauthor multigraph in terms of 24 computer research fields [52]: AI, AIGO, ARC, BIO, CV, DB, DIST, DM, EDU, GRP, HCI, IR, ML, MUL, NLP, NW, OS, PL, RT, SC, SE, SEC, SIM, WWW. A pair of coauthors in this multigraph can have up to 24 parallel edges, each weighted by the number of co-authored papers in one of the 24 computer science fields [65]. Figure 5.3 shows an illustrative example of slices. Consider the example co-author graph in Figure 5.3 (a) with three types of edges: AI, DB and DM, representing the number of coauthored publications on AI conferences (*IJCAI*, *AAAI* and *ECAI*), DB conferences (*SIGMOD*, *VLDB* and *ICDE*), and DM conferences (*KDD*, *ICDM* and *SDM*), respectively. By slice partitioning, we obtain three slices in Figure 5.3 (b), (c) and (d) respectively, one for each category. If we want to compute the coauthor based social influence among researchers in DB and DM area, we only need to perform iterative computation on the coauthor graph using joint publications in DB and DM conferences and journals.



(a) Coauthor Multigraph (b) AI Slice

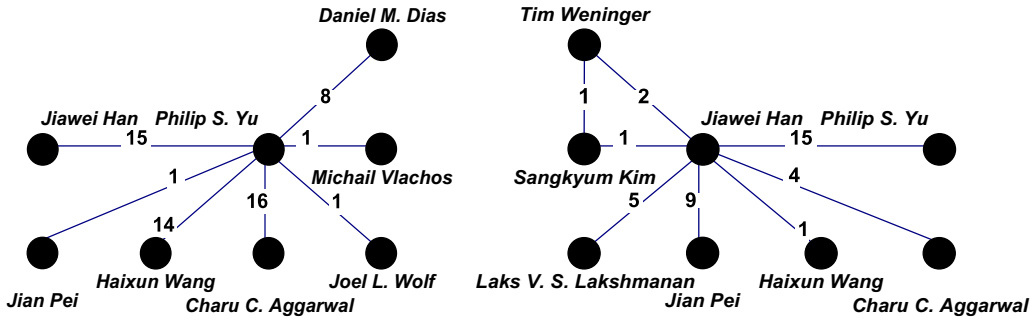
(c) DB Slice (d) DM Slice

Figure 5.3: Slice Partitioning: An Example from DBLP

Another innovative and rewarding usage of slice partitioning is to speed up the iterative graph algorithms on single graphs by performing parallel computation on p slices in parallel. Consider PageRank as an example, the edge weights in the original simple graph are normalized as probabilities in the transition matrix $\mathbf{M}$. Thus, the domain of W is defined on a continuous space over the range $[0, 1]$. In each iteration, PageRank updates the ranking vector $\mathbf{R}$ by iteratively calculating the multiplication between $\mathbf{M}$ and $\mathbf{R}$. However, the transition matrix $\mathbf{M}$ often has skewed edge weight distribution. For instance, the transition matrix for the DBLPS dataset [109] has skewed distribution of transition probabilities (the percentage of edges in the specific weight range to total edges), as shown in Table 5.2. By introducing dimension W, GraphTwist can partition the input DBLPS graph for PageRank into slices along dimension W based on its transition matrix $\mathbf{M}$ and execute the iterative computations at the slice level in parallel. This enables GraphTwist to address skewed edge weight distribution much more efficiently as demonstrated in our experiments reported in Section 5.5, where we set $p = 13, 10, 7, 7, 4$ and 4 for Yahoo, uk-union, uk-2007-05,

Table 5.2: Transition Distribution on DBLPS

e.weight	(0, 0.1]	(0.1, 0.2]	(0.2, 0.3]	(0.3, 0.4]	(0.4, 0.5]	
edges(%)	59.90	18.79	6.99	6.61	5.14	
e.weight	(0.5, 0.6]	(0.6, 0.7]	(0.7, 0.8]	(0.8, 0.9]	(0.9, 1.0)	1.0
edges(%)	0.14	0.21	0.08	0.02	0.00	2.12



(a) Strip 1 (b) Strip 2

Figure 5.4: Strip Partitioning of DB Slice

Twitter, Facebook and DBLPS, respectively. We show that the iterative graph computation using the slice partitioning significantly improves the performance of all graph algorithms we have tested.

The slice partitioning can be viewed as an edge partitioning through clustering by the edge weights. However, when a slice-based subgraph (partition block) together with its intermediate results are too big to fit into the working memory, we further cut the graph into smaller graph parallel units such as dices (see Section 5.3.1) or strips (see Section 5.3.1).

Strip-based Parallel Graph Abstraction

For graphs that are sparse with skewed degree distribution, high degree vertices can incur acute imbalance in parallel computation, and lead to serious performance degradation. This is because the worker thread assigned to a high degree vertex takes much longer time to compute than the parallel threads computing on low degree vertices. To maximize the performance of parallel computation, and ensure better resource utilization and better work balance, in GraphTwist we introduce the strip-based graph partitioning, which cuts a graph along either its source dimension or its destination dimension to obtain strips. Compared

to the dice-based partitioning that cuts a graph (or slices of a graph) along both source and destination dimensions, strips represent larger partition units than dices. A strip can be viewed as a sequence of dices stored physically together. Similarity, we further cut a slice into strips when single slice can not fit into the working memory.

An efficient way to construct in-edge strips (out-edge strips) is to cut an in-edge cube or slice (out-edge cube or slice) along destination (source) dimension D (S). By cutting an in-edge slice of G, J=(S, D, E_J, W_J), along D into q in-edge strips, each strip is denoted as K=(S, D_K, E_K, W_J), where $D_K \subseteq D$ is a subset of destination vertices, and E_K=$\{(u,v)|u \in S, v \in D_K, (u,v).weight \in W_J, (u,v) \in E_J\}$ is a set of directed edges from S to D_K with weights in W_J. An in-edge strip contains all IEDs of a DVP. Similarly, an **out-edge strip** can be defined and it has all OEDs of a SVP. Figure 5.4 gives an illustrative example of strip-based partitioning, where two strips are extracted from the DB slice in Figure 5.3 (c), i.e., all coauthored DB links of *Daniel M. Dias*, *Michail Vlachos* and *Philip S. Yu*, and all coauthored DB links of *Jiawei Han*, *Sangkyum Kim* and *Tim Weninger*. Another important feature of our graph partitioning methods is to choose smaller subgraph blocks such as dice partition or strip partition to balance the parallel computation efficiency among partition blocks and to use larger subgraph blocks such as slice partition or strip partition to maximize sequential access and minimize random access.

Cut-based Parallel Graph Abstraction

A vertex cut is introduced in GraphTwist as a logical partition unit. A dice partition can be viewed as a subgraph composed of multiple vertex cuts, one per vertex. Formally, given an IED H=(S_H, D_H, E_H, W_H), an **out-edge cut** of H is denoted as $L(u)$=(u, D_H, E_L, W_H) where $u \in S_H$ and E_L=$\{(u,v)|v \in D_H, (u,v) \in E_H, (u,v).weight \in W_H\}$. $L(u)$ contains all out-edges of u in this IED. Since edges in each IED are stored by the lexical order of their source vertices, we can easily split an IED into multiple out-edge cuts. Similarly, an OED can be viewed

as a set of **in-edge cut**s, each containing the in-edges between a set of source vertices and a destination vertex. One of the utility-aware pruning optimization at the computation tier is vertex-based pruning based on vertex cuts (see Section 5.4.2 for detail).

Graph Partitioning Algorithms

In the first prototype of GraphTwist, we implement the three parallel graph abstraction based partitioning, placement and access algorithm using a unified graph processing framework with the top-down partitioning strategy. Given a PC, a graph dataset and a graph application, such as PageRank or SSSP, GraphTwist provides the system default partitioning settings on p ($\#Slices$), q ($\#Strips$) and r ($\#Dices$). We defer the detailed discussion on the settings of optimal partitioning parameters to Section 5.3.5. Based on the system-supplied default settings of the partitioning parameters, we first partition a graph into p slices, and then we partition each slice into q strips, and partition each strip into r dices. The partitioning parameters are chosen such that each graph partition block and its intermediate results will fit into the working memory. In GraphTwist, we first partition the source vertices of a graph into SVPs, partition destination vertices of the graph into DVPs and then partition the graph into edge partitions slice by slice, strip by strip or dice by dice. Figure 5.2 gives an example of vertex partitioning and edge partitioning for the graph in Figure 5.1. Based on the system-recommended partitioning parameter, we store the graph in the physical storage using the smallest partition unit given by the system configuration. Dice is the smallest and most frequently used partition block in GraphTwist.

GraphTwist provides a three-level partition-index structure to access dice partition blocks on disk slice by slice, strip by strip or dice by dice. When the graph application has sufficient memory to host the entire index in memory, sequential access to dices stored in physical storage can be maximized. For example, PageRank algorithm requires computing the ranking score for every vertex using its incoming edges and its corresponding source vertices in each iteration. Thus, the PageRank implementation in GraphTwist will start to

Algorithm 4 GraphPartitioning($G, app, p, q, r, flag$)

```
1: Sort source vertices by the lexical order of vertex IDs;
2: Sort destination vertices by the lexical order of vertex IDs;
3: Sort edges by source vertex ID, destination vertex ID and edge weight;
4: switch(flag)
5:    case 0: select p, q, r input by user;
6:    case 1: detect resource, calculate p, q, r for app online;
7:    case 2: select the optimal p, q, r with offline learning;
8: Divide Winto p intervals;
9: Split G into p in-edge slices;
10: Divide D into p DVPs; //partition destination vertices
11: Split p in-edge slices into p×q in-edge strips; //partition edges in each in-edge slice
    into q strip-based edge partitions
12: divide S into q SVPs; //partition source vertices
13: Split p×q in-edge strips into p×q×r IEDs; //partition edges in each in-edge strip into
    r dice-based edge partitions
14: Compress DVPs, SVPs and IEDs, and write them back to disk; //compress vertex par-
    titions and edge partitions
15: Build the indices for p slices, p×q strips and p×q×r IEDs;
```

access the graph by one in-edge strip at a time. For each strip, we check if there are multiple slices corresponding to this strip, For each strip and a corresponding slice, we access the dices corresponding to the strip and the slice. GraphTwist provides a function library to support various iterative graph applications with a conventional vertex-oriented programming model. Algorithm 4 provides the pseudo code for an example function GraphPartitioning. Given a graph G and a graph application, say PageRank, it constructs an in-edge cube representation of G and partitions its in-edge cube through slice, strip and dice abstractions. We do not build the out-edge cube of graph since PageRank does not use outgoing edges.

5.3.2 Access Locality Optimization

We describe two access locality based optimizations implemented in the first prototype of GraphTwist: (1) graph index structure for indexing slices, strips and dices; and (2) graph partition-level compression for optimizing disk I/Os.

Partition Index. GraphTwist stores a graph G in the physical storage as either dices or strips or slices. The iterative graph computation is performed in parallel at the partition

level, be it a dice or a strip or a slice. Each partition block corresponds to a subgraph of G. In order to provide fast access to partition blocks of G stored on disk using slice or strip or dice specific conditions, and to ensure that each partition subgraph is loaded into the working memory only once or minimum number of times in each iteration, we design a general graph index structure to enable us to construct the slice index, the strip index or the dice index. The dice index is a dense index that maps a dice ID and its DVP (or SVP) to the chunks on disk where the corresponding dice partition is stored physically. The strip index is a two-level sparse index, which maps a strip ID to the dice index blocks and then map each dice ID to the dice partition chunks in the physical storage. Similarly, the slice index is a three-level sparse index with slice index blocks at the top, strip index blocks at the middle and dice index blocks at the bottom, enabling fast retrieval of dices with a slice-specific condition. In addition, we also maintain a vertex index that maps each vertex to the set of subgraph partitions containing this vertex, as shown in Figure 5.2. This index allows fast lookup of the partitions relevant to a given vertex.

Partition-level Compression. It is known that iterative computations on large graphs incur non-trivial cost for the I/O processing. For example, the I/O processing of Twitter dataset on a PC with 4 CPU cores and 16GB memory takes 50.2% of the total running time for PageRank (5 iterations). In addition to utilize index, we employ partition-level compression to increase the disk I/O efficiency. Concretely, GraphTwist transforms the raw graph data into partition blocks and applies in-memory gzip compression to transform each partition block into a compressed format before storing them on disk. We maintain two buffers in memory, one for input graph data and another for in-memory compressed output graph data. As soon as a sequential read into the input stream buffer is completed, we start the in-memory gzip compression and append the compressed data to the output stream buffer. After finishing the compression, GraphTwist sequentially writes the compressed chunks in the output stream buffer to disk. This one-time compression cost at the building time can provide quick access to stored graph partitions and reduce I/O time in each of

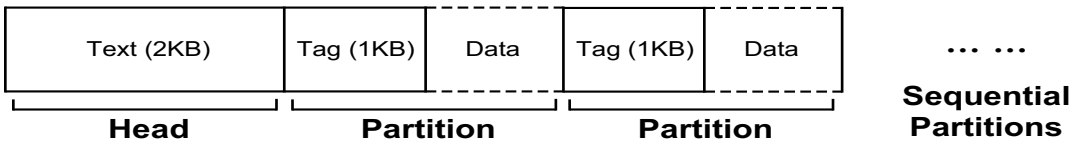


Figure 5.5: Data Structure for Compressed Graph

the graph computation iteration. A gzip-compressed graph file consists of a 2KB head section and multiple partition sections, as shown in Figure 5.5. The head section stocks the descriptive information about the graph: the number of vertices, the number of edges, the edge type (undirected or directed), the domain of edge weights, the average degree, and the maximum degree. Each partition section consists of a 1KB metadata tag followed by the data in the partition. The tag provides the descriptive information about the specific partition: the data type, the size of partition, the number of edges, the source range, the destination range, the domain of edge weights.

5.3.3 Programmable GraphTwist Interface

GraphTwist provides a conventional programming interface to enable users to write their iterative graph algorithms using the vertex centric computation model. By supporting the vertex centric programming API, the users of GraphTwist only need to provide their iterative algorithms in terms of vertex-level computation using the functions provided in our API, such as Scatter and Gather. For each iterative graph algorithm defined by users using our API, GraphTwist will compile it into a sequence of GraphTwist internal function (routine) calls that understand the internal data structures for accessing the graph by subgraph partition blocks. These routines can carry out the iterative computation for the input graph either slice by slice, strip by strip, or dice by dice. For example, PageRank algorithm can be written by simply proving the computation tasks, as shown in Algorithm 5.

For each vertex in a DVP to be updated, GraphTwist maintains a temporary local buffer to aggregate received messages. Since each vertex v may contain both in-edges and out-edges, users only need to define two application-level functions to instruct the system-level

Algorithm 5 PageRank

```
1:  Initialize(v)
2:    v.rank = 1.0;
3:
4:  Scatter(v)
5:    msg = v.rank/v.degree;
6:    //send msg to destination vertices of v's out-edges
7:
8:  Gather(v)
9:    state = 0;
10:   for each msg of v
11:   //receive msg from source vertices of v's in-edges
12:     state += msg; //summarize partial vertex updates
13:   v.rank = 0.15 + 0.85 * state; //produce complete vertex update
```

scatter and gather routines to perform the concrete aggregations. Given a vertex, the Scatter function works on a selection of v's neighbors, say the destination vertices of the out-edges of v, to scatter its update based on its vertex state from the previous iteration. Similarly, the Gather function works on a selection of v's neighbors, e.g., the source vertices of the in-edges of v, to gather the information in order to update its vertex state, and pass this updated vertex state to the next iteration if the update commits and otherwise assign a partial commit to the gather task.

Vertices are accessed either by DVP or SVP depending on the specific graph algorithm. For each vertex to be updated, GraphTwist maintains a temporary local buffer to aggregate received messages. We implement the Scatter API function as follows: First, an internal routine called GraphScan is invoked, which will access the graph dataset on disk by partition blocks (say dices). Then the PartitionParallel routine will be invoked to assign multiple threads to process multiple partitions in parallel, one thread per partition subgraph block. For each partition subgraph, the VertexParallel routine is called to execute multiple subthreads in parallel, one per vertex. At each vertex thread, the Scatter routine is performed. Given that the Scatter function at the system level intends to send the vertex state of v to all or a selection of its (out-edge) neighbor vertices, by utilizing the vertex-partition map (see Figure 2), each vertex thread will check if all partition blocks containing v as a source

vertex have been processed. If so, then v finishes its scatter task with a commit status; Otherwise, v registers a partial commit.

Similarly, the Gather function in our API will be carried out by executing a sequence of four tasks with the first three routines identical to the implementation of the scatter function. The fourth routine is the Gather routine, which executes the gather task in two phases: (1) intra-partition gather, performing partial update of vertices or edges within a partition subgraph block, and (2) cross-partition gather, combining partial updates from multiple partition blocks. We call the vertices that belong to more than one subgraph partitions (e.g., dices) the border vertices. The cross-partition gather is only necessary for the border vertices. The Gather routine first records the messages that v has received from the source vertices of v's in-edges in the receive buffer, produces the count of the received messages, combines the update messages in the receive buffer using the local aggregation operation provided in the user-defined gather function, and then store the partial update as the new vertex state of v. At the end of the aggregation operation, if the received message count is the same as the in-degree of v, then we get the final update of v, and store it as the new state of v for this iteration. Otherwise, if the received message count is less than the in-degree of v, we mark this vertex as a border vertex, indicating that it belongs to more than one partition blocks and thus needs to enter the cross-partition gather phase. In the next subsection, we will discuss how GraphTwist executes the cross-partition gather task to combine the partial updates from different edge partitions at different levels of granularity to generate the complete update.

5.3.4 Synchronization and Multi-threading

To maximize parallelism in iterative graph computations, GraphTwist provides parallel processing at two levels: (1) parallel processing at the subgraph partition level (slice, strip or dice), and (2) parallel processing at the vertex level. A number of design choices are made carefully to support effective synchronization and multi-threading, ensuring vertex

and edge update consistency.

Parallel partial vertex update. As vertices in different dice or strip based subgraph partitions belong to different DVPs (or SVPs), and the DVPs (SVPs) from different strip or dice partitions are disjoint within the given graph or a slice of the graph, the vertex update can be executed safely in parallel on multiple strip or dice based partitions, providing partition-level graph parallelism. Furthermore, the vertex update can also be executed safely in parallel on multiple vertices within each strip or dice since each vertex within the same DVP (or SVP) is unique. Thus, GraphTwist implements the two-level graph computation parallelism at the partition level and at the vertex level.

However, all the parallel vertex updates at both partition-level and vertex level are partial for two reasons: (1) although the edge sets in different in-edge subgraph partitions are disjoint, an edge may belong to one in-edge partition and one out-edge partition for strip or dice based partitions; thus concurrent edge update needs to be synchronized; (2) a vertex may belong to more than one partitions. Thus, the vertex updates performed within a strip or dice partition are partial and need to do cross-partition gather among strip or dice partitions; and (3) the associated edges of a DVP (or SVP) may lie in multiple slices. Thus, the vertex updates performed concurrently on strip or dice partitions within each slice are partial and need to do cross-partition gather among slices.

For cross-partition gather at slice level, for each vertex, there are at most p partial vertex update states, one per slice. We need to aggregate all the partial vertex update states for each vertex to obtain its final vertex update state before moving to the next iteration. To ensure the correctness and consistency of obtaining the final vertex updates via aggregating such partial vertex update states, during each iteration, GraphTwist uses an individual thread to sequentially aggregate all partial vertex updates of a single DVP (or SVP) slice by slice.

Similarly, for cross-partition gather at strip or dice level, GraphTwist divides the strip or dice based partitions that share the same DVP (or SVP) into DVP (or SVP) specific partition groups, and sets the number of DVPs (or SVPs) to be processed in parallel by

the number of concurrent threads used (*#threads*) such that an individual memory block, i.e., partial update list is assigned to each partition group for cross-partition gather. Now each of the individual threads is dedicated to each edge partition within the DVP (or SVP) specific partition groups and execute one such partition at a time. In order to avoid conflict, GraphTwist maintains a *counter* with an initial value of 0 for each specific partition group. When a thread finished the Scatter process of an edge partition within the partition group: put the partial update into the partial update list, this thread checks if *counter* is equal to the number of associated edge partitions for the specific partition group. If not, this thread performs *counter*++ and the scheduler assigns an unprocessed edge partition within the same specific partition group to it. Otherwise, we know that GraphTwist have finished the processing of all edge partitions within the partition group. Thus, this thread continues to perform the Gather process to aggregate all partial updates of this DVP (or SVP) in its partial update list to generate its complete update. Finally, the final update of this DVP (or SVP) in the current iteration are written back to disk. Then, this thread will start to fetch and process the next unfinished or unprocessed DVP (or SVP) and the set of subgraph partitions associated to this DVP (or SVP) in the same manner. We complete one round of iterative computation when vertices in all DVPs (or SVPs) are examined and updated.

Parallel edge update. In GraphTwist each edge must belong to one in-edge dice (IED) and one out-edge dice (OED). Thus, edge update is relatively straightforward. Gra- phLego also implements a two-level parallelism at the strip level by strip threads and at the vertex level by vertex subthreads. An individual strip thread is assigned to a single DVP (or SVP) to sequentially execute the updates of associated edges of this DVP (or SVP) slice by slice. When a DVP (or SVP) thread finishes the updates of all associated edges of a DVP (or SVP), this DVP (or SVP) thread will fetch and process the associated edges of the next unprocessed DVP (or SVP) in the same manner without synchronization.

Figure 5.6 presents an example of partial update at the partition level. There are k available threads $T_1, \cdots, T_k$ in the system and m associated edge partitions $P_1, \cdots, P_m$ in the

partition group of SVP_j. One thread processes only one edge partition at a time. We classify vertices in each partition into three categories: (1) internal vertices belonging to only one partition; (2) border vertices belonging to multiple partitions but their vertex updates are confined to only one partition by the given graph algorithm; and (3) critical border vertices belonging to multiple partitions and their vertex updates depends on multiple partitions. In the example of Figure 5.2, given the graph application of PageRank, which only uses in-edges, the partition group of DVP_1 contains only one dice partition P_1. v_2 and v_4 are internal vertices, and v_3, v_5 and v_6 are border vertices. Similarly, the partition group of DVP_2 contains only one dice partition P_3, v_7, v_8, v_9, and v_{10} are internal vertices, and v_{11} is border vertex. The partition group of DVP_3 contains two dice partitions: P_2 and P_4. Vertex v_{14} is internal vertex, and v_{13} and v_{15} are border vertices for P_2. Vertex v_{16} is internal vertex for P_4. Vertex v_{12} is the only critical border vertex for both P_2 and P_4, because only v_{12} receives partial updates from both partitions.

For internal vertices and border vertices, we can commit their vertex updates upon the completion of the partition-level parallel computation in each iteration. However, for critical border vertices (e.g. v_i in Figure 5.6), its associated edges may distribute into multiple edge partitions, say $P_1, \cdots, P_m$. In order to avoid conflict, GraphTwist maintains a partial update list with an initial *counter* of 0 for each critical border vertex in memory. If a thread T_x ($1 \leq x \leq k$) processes an edge partition P_y ($1 \leq y \leq m$) and needs to update the state of v_i, T_x first executes the Scatter process to put the partial update $v_{i,y}$ by P_y (a temporary local copy of vertex v_i) into the partial update list, and then check if *counter* is equal to the number of v_i's associated edge partitions from the vertex map, as shown in Figure 5.2. If not, T_x performs *counter*++ and the scheduler assigns an unprocessed edge partition to T_x. Otherwise, we know that P_y is the last processed edge partitions (e.g. P_m in Figure 5.6). Thus, T_x continues to perform the Gather process to aggregate all partial updates of v_i in its partial update list to generate a complete update of v_i.

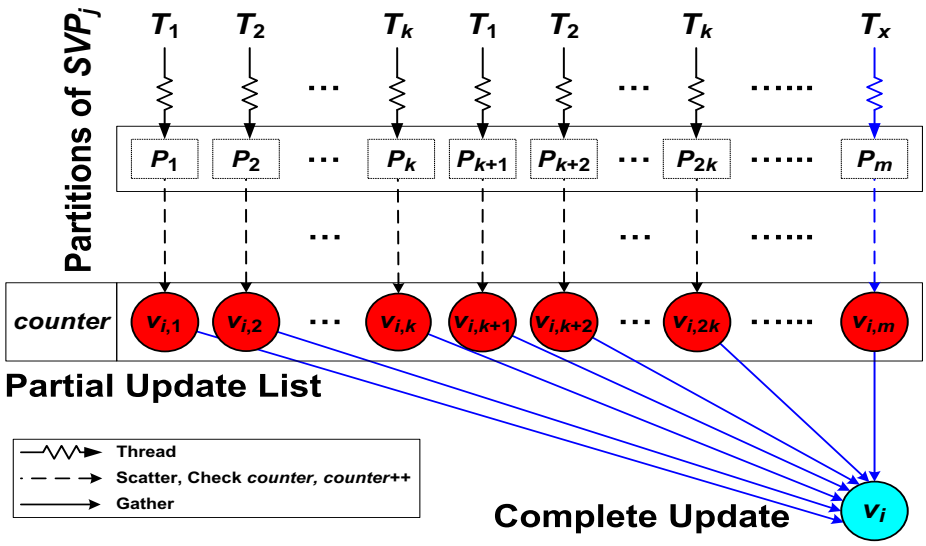


Figure 5.6: Multithreading Update and Asynchronization

5.3.5 Configuration of Partitioning Parameters

Given a total amount of memory available, we need to determine the best settings of the partitioning parameters for achieving the optimal computational performance. GraphTwist supports three alternative methods to determine the settings of parameters: user definition, simple estimation, and regression-learning based configurations.

User Definition. We provide user-defined configuration as an option for expert users to modify the system default configuration.

Simple Estimation. The general heuristic used in simple estimation is to determine p ($\#Slices$), q ($\#Strips$) and r ($\#Dices$) based on the estimation of whether each subgraph block for the given partition unit plus the intermediate results will fit into the available working memory. In GraphTwist, we provide simple estimation from two dimensions: the past knowledge from regression-based learning and the simple estimation in the absence of prior experiences by estimating the size of the subgraph blocks and the intermediate results depending on the specific graph applications. GraphTwist uses the parameter settings produced by simple estimation as the system-defined default configuration.

In summary, the decision of whether to use slice, strip or dice as the partition unit to access the graph data on disk and to process the graph data in memory should be based

on achieving a good balance between the following two criteria: (1) We need to choose the partition unit that can best balance the parallel computation workloads with bounded working memory; and (2) we need to minimize excessive disk I/O cost by maximizing sequential disk access in each iteration of the graph algorithm.

Regression-based Learning. A number of factors may impact the performance of GraphTwist, such as concrete applications, graph datasets, the number of CPU cores, the DRAM capacity. Thus, for a given graph application, a given dataset and a given server, we want to find the latent relationship between the number of partitions and the runtime. In order to learn the best settings of these partitioning parameters, we first utilize multiple polynomial regression [111] to model the nonlinear relationship between independent variables p, q or r and dependent variable T (the runtime) as an n^{th} order polynomial. A regression model relates T to a function of p, q, r, and the undetermined coefficients α: $T \approx f(p,q,r,\alpha) = \sum_{i=1}^{n_p}\sum_{j=1}^{n_q}\sum_{k=1}^{n_r}\alpha_{ijk}p^iq^jr^k + \epsilon$ where n_p, n_q and n_r are the highest orders of variables p, q or r, and ϵ represents the error term of the model.

We then select m samples of (p_l, q_l, r_l, T_l) $(1 \le l \le m)$ from the existing experiment results, such as the points in Figure 5.18 (c)-(d), to generate the following m linear equations:

$$\begin{aligned} T_1 &= \sum_{i=1}^{n_p}\sum_{j=1}^{n_q}\sum_{k=1}^{n_r}\alpha_{ijk}p_1^iq_1^jr_1^k + \epsilon \\ &\cdots \qquad \cdots \\ T_m &= \sum_{i=1}^{n_p}\sum_{j=1}^{n_q}\sum_{k=1}^{n_r}\alpha_{ijk}p_m^iq_m^jr_m^k + \epsilon \end{aligned} \tag{5.1}$$

We adopt the least squares approach [112] to solve the above overdetermined linear equations and generate the regression coefficients α_{ijk}. Finally, we utilize a successive convex approximation method (SCA) [113] to solve this polynomial programming problem with the objective of minimizing the predicted runtime and generate the optimal p, q and r. The experimental evaluation demonstrates that our regression-based learning method can select the optimal setting for the partitioning parameters, which gives GraphTwist the best

performance under the available system resource.

5.4 Computation-tier Optimization

Many large-scale real-world graphs have millions of vertices and billions of edges. Handling such big graphs in memory may require tens or hundreds of gigabytes of DRAM. Popular iterative graph applications usually consist of a series of matrix-vector computations [96, 99, 65, 114] or matrix-matrix computations [97, 115, 20, 14, 28, 51]. Besides processing graphs partition by partition, another way to address this problem is to speed up the iterative computations by pruning some insignificant vertices or edges based on a certain statistical measure. For example, PageRank needs to iteratively calculate the multiplication between the transition matrix $\mathbf{M}$ and the ranking vector $\mathbf{R}$. In GraphTwist, we set rank entries in $\mathbf{R}$ as vertex states and elements in $\mathbf{M}$ as edge weights. Given that the core utility of PageRank is to produce an ordered list of vertices by their PageRank scores, we want to prune some insignificant edges to speed up the PageRank computation while preserving the ranking order of all vertices. Figure 5.7 presents the transition matrix $\mathbf{M}$ of a given graph with 0/1 edge weights. According to the edge weight distribution in $\mathbf{M}$, we partition $\mathbf{M}$ into two slices: $\mathbf{M_1}$ with edge weights of $[0.3, 1]$ and $\mathbf{M_2}$ with edge weights of $(0, 0.3)$. Given an initial ranking vector $\mathbf{R}(0)$ and a damping factor d, we utilize the transition matrix $\mathbf{M}$ to calculate the exact ranking vector $\mathbf{R}(1)$ and use the selected slice $\mathbf{M_1}$ to compute the approximate ranking vector $\mathbf{R_1}(1)$ since the pruned edges in $\mathbf{M_2}$ have relatively small weights. Although the corresponding ranking scores in $\mathbf{R}(1)$ and $\mathbf{R_1}(1)$ are somewhat different, the ranking orders in the two ranking vectors are identical, indicating that the pruning of slice $\mathbf{M_2}$ preserves the utility of PageRank.

This motivates us to propose two utility-aware pruning techniques: slice pruning and cut pruning, to speed up the iterative graph computations while preserving the computational utility. Given that matrix multiplication is the fundamental core for many iterative graph applications, we use the matrix multiplication as an example to describe two pruning

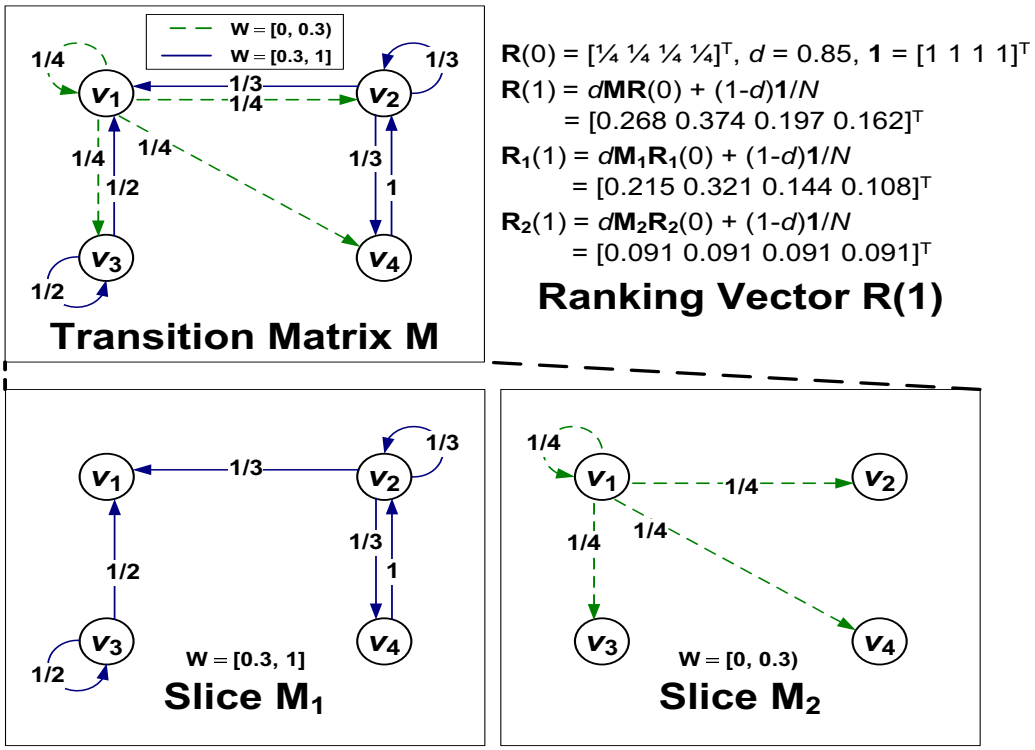


Figure 5.7: PageRank with Slice Pruning

techniques: (1) speed up matrix multiplication by pruning insignificant slices with provable error bound, and (2) accelerate strip multiplication by pruning trivial vertex cuts in the strips with bounded error.

5.4.1 Slice Pruning (Subgraph-based Pruning)

The main idea is to reduce the computational cost by pruning some insignificant slices based on a statistical measure while preserving the utility of the graph computation. To speed up the iterative computations, one intuitive idea is to prune those sparse slices with small weights. We introduce the concept of slice density as a statistical measure to evaluate the importance of a slice. For presentation convenience, we use a symbol to represent a graph partition and its matrix (or vector) representation interchangeably when no confusion occurs. Cube, slice, strip and dice correspond to matrices, and cut corresponds to vector.

Definition 19 (Slice Density) *Let $I=(S, D, E, W)$ be a 3D cube of a directed graph $G=(V, E, W)$, and $J=(S, D, E_J, W_J)$ be a slice of I where $S=V$, $D=V$, $W_J \subseteq W$, and $E_J=\{(u, v)|u \in S, v \in D,$*

$(u, v).weight \in W_J, (u, v) \in E\}$*, the density of* J *is defined as follow.*

$$SliceDensity(J) = \|J\|_F = \sqrt{\sum_{u=1}^{n} \sum_{v=1}^{n} \sum_{(u,v)\in E_J} w(u, v)^2} \tag{5.2}$$

where J *denotes the slice itself as well as its matrix representation,* F *is the Frobenius norm, and* $w(u, v)$ *represents the weight of an edge* (u, v)*. If a slice is dense and has large edge weights, then it will have a large density.*

Next we illustrate how to use the *SliceDensity* measure for matrix multiplication with slice pruning to prune the sparse slices with small weights. Given two matrices A consisting of s out-edge slices $A_1, \cdots, A_s$ and B comprising t in-edge slices $B_1, \cdots, B_t$, we decompose a matrix multiplication $C{=}A{\times}B$ into $s{*}t$ slice multiplications, i.e., $C{=}\sum_{k=1}^{s}\sum_{l=1}^{t} A_k{\times}B_l$. We propose a Monte-Carlo algorithm to compute a multiplication approximation: choose "important" $x{*}y$ ($<< s{*}t$) of $s{*}t$ slice multiplications to calculate the multiplication, while the expectation of the approximate multiplication is equal to the exact multiplication. Slice pruning strategy is two-fold: (1) pick slices according to the amount of "information" the slices contain; (2) rescale the multiplication to compensate for the slices that are not picked.

Concretely, we first compute the selection probability ρ_{kl} of each pair of slices (A_k and B_l) in terms of their precomputed *SliceDensity* values. Then we independently do $x{*}y$ selection trials and choose $x{*}y$ slice pairs from the original $s{*}t$ slice pairs in terms of selection probabilities. The indices of extracted slices of A and B in the h^{th} trial are kept in $\Phi(h)$ and $\Psi(h)$ respectively. GraphTwist implements the slice multiplication at the strip level by invoking the **StripMultiply** method in Algorithm 7. We also rescale the edges in F with rescaling factor $\frac{1}{x{*}y{*}\rho_{\Phi(h)\Psi(h)}}$ to compensate for the slices that are not picked, where $\rho_{\Phi(h)\Psi(h)}$ denotes the selection probability of the $\Phi(h)^{th}$ slice of A and the $\Psi(h)^{th}$ slice of B. Thus, the multiplication approximation is calculated as $\tilde{C}{=}\sum_{h=1}^{x*y} \frac{A_{\Phi(h)}{\times}B_{\Psi(h)}}{x{*}y{*}\rho_{\Phi(h)\Psi(h)}}$. The edges can be safely updated in parallel at strip level since F (or E) corresponds to a unique IED and a unique OED of $\tilde{C}$ such that the IEDs (or OEDs) generated by different strip multiplications

Algorithm 6 MatrixMultiply(A, B, x, y, q, z)

Initialize $E[1 \cdots \lfloor n/q \rfloor][1 \cdots \lfloor n/q \rfloor]$;
for $k = 1, \cdots, s$
  for $l = 1, \cdots, t$
    $\rho_{kl} = \frac{SliceDensity(A_k) * SliceDensity(B_l)}{\sum_{i=1}^{s} \sum_{j=1}^{t} SliceDensity(A_i) * SliceDensity(B_j)}$;
for $h = 1, \cdots, x*y$ independently
  Pick $a, b, a \in \{1, \cdots, s\}, b \in \{1, \cdots, t\}$ with Prob$(a = k, b = l) = \rho_{kl}, k = 1, \cdots, s, l = 1, \cdots, t$;
  $\Phi = \Phi \cup \{a\}$; $\Psi = \Psi \cup \{b\}$;
parallel for $i = 1, \cdots, q$
  parallel for $j = 1, \cdots, q$
    for $h = 1, \cdots, x*y$
      $F =$ **StripMultiply**$(A, \Phi(h), i, B, \Psi(h), j, z)$;
      parallel for each $e \in F$
        $E[e.source][e.destination].AddWeight(e.weight/(x*y*\rho_{\Phi(h)\Psi(h)}))$;
  Split E into p partitions in terms of weight distribution;
  Write $IED[j][i]$ in p in-edge slices of $\tilde{C}$ to disk;
  Write $OED[i][j]$ in p out-edge slices of $\tilde{C}$ to disk;

in the same slice pair are irrelevant to each other.

The following theorems state that our slice pruning strategy can achieve a good approximation with bounded error.

Theorem 18 *Given an actual multiplication* $C = \sum_{k=1}^{s} \sum_{l=1}^{t} A_k \times B_l$*, a multiplication approximation* $\tilde{C} = \sum_{h=1}^{x*y} \frac{A_{\Phi(h)} \times B_{\Psi(h)}}{x*y*\rho_{\Phi(h)\Psi(h)}}$*, then the expectation* $E(\tilde{C}(i, j)) = C(i, j)$ *for* $\forall i, j \in \{1, \cdots, n\}$.

Proof. Suppose that a random variable $X_h = \frac{A_{\Phi(h)} \times B_{\Psi(h)}}{x*y*\rho_{\Phi(h)\Psi(h)}}(i, j) = \sum_{m=1}^{n} \frac{A_{\Phi(h)}(i,m) * B_{\Psi(h)}(m,j)}{x*y*\rho_{\Phi(h)\Psi(h)}}$ *where* $h \in \{1, \cdots, x * y\}$. X_h *represents the entry in the* i^{th} *row and* j^{th} *column of multiplication between slice* $A_{\Phi(h)}$ *and slice* $B_{\Psi(h)}$ *with edge rescaling of* $\frac{1}{x*y*\rho_{\Phi(h)\Psi(h)}}$*, then all of* X_h*s are independent random variables. Also,* $\tilde{C}(i, j) = \sum_{h=1}^{x*y} X_h$*. Thus,* $E(\tilde{C}(i, j)) = E(\sum_{h=1}^{x*y} X_h) = \sum_{h=1}^{x*y} E(X_h)$ *since all of* X_h*s are defined on the same probability space. In addition,* $E(X_h) = \sum_{k=1}^{s} \sum_{l=1}^{t} \rho_{kl} \sum_{m=1}^{n} \frac{A_k(i,m) * B_l(m,j)}{x*y*\rho_{kl}}$ $= \sum_{k=1}^{s} \sum_{l=1}^{t} \sum_{m=1}^{n} \frac{A_k(i,m) * B_l(m,j)}{x*y} = \frac{1}{x*y} C(i, j)$*. In summary,* $E(\tilde{C}(i,j)) = \sum_{h=1}^{x*y} E(X_h) = \sum_{h=1}^{x*y} \frac{1}{x*y} C(i, j) = C(i, j)$.

Theorem 19 *Given the same definitions in Theorem 24, then the variance* $Var(\tilde{C}(i, j)) = \sum_{k=1}^{s} \sum_{l=1}^{t} \frac{1}{x*y*\rho_{kl}} (\sum_{m=1}^{n} A_k(i, m) * B_l(m, j))^2 - \frac{1}{x*y} C(i, j)^2$ *for* $\forall i, j \in \{1, \cdots, n\}$.

Proof. Since $\tilde{C}(i,j)$ *is the sum of* $x*y$ *independent random variables, i.e.,* $\tilde{C}(i,j)=\sum_{h=1}^{x*y} X_h$, *then* $Var(\tilde{C}(i,j))=Var(\sum_{h=1}^{x*y} X_h)=\sum_{h=1}^{x*y} Var(X_h)$. *Also,* $Var(X_h)=E(X_h^2)-E^2(X_h)=\sum_{k=1}^{s}\sum_{l=1}^{t}\rho_{kl}(\sum_{m=1}^{n}\frac{A_k(i,m)*B_l(m,j)}{x*y*\rho_{kl}})^2 -(\frac{1}{x*y}C(i,j))^2=\sum_{k=1}^{s}\sum_{l=1}^{t}\frac{1}{x^2*y^2\rho_{kl}}(\sum_{m=1}^{n}A_k(i,m)*B_l(m,j))^2-\frac{1}{x^2*y^2}C(i,j)^2$. *Thus,* $Var(\tilde{C}(i,j))=\sum_{h=1}^{x*y}(\sum_{k=1}^{s}\sum_{l=1}^{t}\frac{1}{x^2*y^2*\rho_{kl}}(\sum_{m=1}^{n}A_k(i,m)*B_l(m,j))^2-\frac{1}{x^2*y^2}C(i,j)^2) = \sum_{k=1}^{s}\sum_{l=1}^{t}\frac{1}{x*y*\rho_{kl}}(\sum_{m=1}^{n}A_k(i,m)*B_l(m,j))^2-\frac{1}{x*y}C(i,j)^2$.

Theorem 20 *Given the same definitions in Theorem 24,*

$$\min_{\rho_{kl}}\left\{E(\|\bar{C}-C\|_F^2):\sum_{k=1}^{s}\sum_{l=1}^{t}\rho_{kl}=1\right\}=\frac{1}{x*y}(\sum_{k=1}^{s}\sum_{l=1}^{t}\|A_kB_l\|_F)^2 - \frac{1}{x*y}\|C\|_F^2 \text{ when } \rho_{kl}=\frac{\|A_kB_l\|_F}{\sum_{k=1}^{s}\sum_{l=1}^{t}\|A_kB_l\|_F} \tag{5.3}$$

Proof. According to Theorem 24, $E(\|\tilde{C}-C\|_F^2)=\sum_{i=1}^{n}\sum_{j=1}^{n}E((\tilde{C}(i,j)-C(i,j))^2)=\sum_{i=1}^{n}\sum_{j=1}^{n}(E(\tilde{C}(i,j)^2)-E^2(\tilde{C}(i,j)))=\sum_{i=1}^{n}\sum_{j=1}^{n}Var(\tilde{C}(i,j))$ *since* $E(\tilde{C}(i,j))=C(i,j)=E(C(i,j))$. *From Theorem 19,* $\sum_{i=1}^{n}\sum_{j=1}^{n}Var(\tilde{C}(i,j))=\sum_{i=1}^{n}\sum_{j=1}^{n}(\sum_{k=1}^{s}\sum_{l=1}^{t}\frac{1}{x*y*\rho_{kl}}(\sum_{m=1}^{n}A_k(i,m)*B_l(m,j))^2-\frac{1}{x*y}C(i,j)^2)=\frac{1}{x*y}\sum_{k=1}^{s}\sum_{l=1}^{t}\frac{1}{\rho_{kl}}\|A_kB_l\|_F^2 -\frac{1}{x*y}\|C\|_F^2$.

Let $f(\rho_{kl})=\frac{1}{x*y}\sum_{k=1}^{s}\sum_{l=1}^{t}\frac{1}{\rho_{kl}}\|A_kB_l\|_F^2-\frac{1}{x*y}\|C\|_F^2$, *non-zero entries in Hessian* $f''(\rho_{kl})$ *are diagonal entries* $\frac{2}{x*y*\rho_{kl}^3}\|A_kB_l\|_F^2$ *such that* $h^Tf''(\rho_{kl})h\geq 0$ *for* $\forall st\times 1$*-vector* h *in* $\mathbb{R}^{st}$, *i.e.,* $f''(\rho_{kl})$ *is positive-semidefinite.* $f(\rho_{kl})$ *is thus a convex function. In addition, the constraint set* $\{\rho_{kl}|\sum_{k=1}^{s}\sum_{l=1}^{t}\rho_{kl}=1\}$ *is both convex and concave. The minimization problem of a convex function on a convex set is a convex programming problem. Thus, there exists a verifiable sufficient and necessary condition for global optimality. Let* $\rho=[\rho_{11};\cdots;\rho_{1t};\rho_{21};\cdots;\rho_{2t};\cdots;$

$\rho_{s1}; \cdots; \rho_{st}]$, *we calculate the global optimal solution by solving the KKT condition.*

$$\nabla_{\rho}\Big(\frac{1}{x*y}\sum_{k=1}^{s}\sum_{l=1}^{t}\frac{1}{\rho_{kl}}\|A_kB_l\|_F^2 - \frac{1}{x*y}\|C\|_F^2 + \lambda(\sum_{k=1}^{s}\sum_{l=1}^{t}\rho_{kl} - 1)\Big) = 0$$
$$\sum_{k=1}^{s}\sum_{l=1}^{t}\rho_{kl} - 1 = 0 \tag{5.4}$$

We get $\lambda=\frac{1}{x*y}(\sum_{k=1}^{s}\sum_{l=1}^{t}\|A_kB_l\|_F)^2$ *and the optimal solution* $\rho_{kl}=\frac{\|A_kB_l\|_F}{\sum_{k=1}^{s}\sum_{l=1}^{t}\|A_kB_l\|_F}$. *Therefore, the optimal value of* $E(\|\tilde{C}-C\|_F^2)$ *is equal to* $\frac{1}{x*y}(\sum_{k=1}^{s}\sum_{l=1}^{t}\|A_kB_l\|_F)^2-\frac{1}{x*y}\|C\|_F^2$.

However, A_kB_l represents the multiplication between slices A_k and B_l. Thus, we can not get such ρ_{kl} before doing slice multiplication. Notice that $\|A_kB_l\|_F^2=\sum_{i=1}^{n}\|A_kB_l(i,:)\|_2^2=\sum_{i=1}^{n}\|A_k(i,:)B_l\|_2^2\leq\sum_{i=1}^{n}\|A_k(i,:)\|_2^2\|B_l\|_F^2=\|A_k\|_F^2\|B_l\|_F^2\Rightarrow\|A_kB_l\|_F\leq\|A_k\|_F\|B_l\|_F$ where $A_kB_l(i,:)$ represents the i^{th} row of multiplication between two slices, and $A_k(i,:)$ denotes the i^{th} row of A_k. Thus, we use the *SliceDensity* measure in Eq.(8.4) to generate a near-optimal $\rho_{kl}=\frac{\|A_k\|_F\|B_l\|_F}{\sum_{i=1}^{s}\sum_{j=1}^{t}\|A_i\|_F\|B_j\|_F}$ in line 4 in Algorithm 6.

Figure 5.8 presents an example of slice pruning for computing the square of the transition matrix $\mathbf{A}$ of a given graph. We partition $\mathbf{A}$ into two equal-size slices: $\mathbf{A_1}$ with edge weights of $[0.25, 1]$ and $\mathbf{A_2}$ with edge weights of $[0, 0.25)$. The exact matrix power $\mathbf{A}^2$ is equal to $\sum_{k=1}^{2}\sum_{l=1}^{2}A_k\times A_l$. If we want to use only one slice multiplication to approximate $\mathbf{A}^2$, then it is very probable that $\mathbf{A_1}\times\mathbf{A_1}$ is picked to approximate $\mathbf{A}^2$ with rescaling factor $\frac{1}{x*y*\rho_{11}}$ since ρ_{11} is maximal.

5.4.2 Cut Pruning (Vertex-based Pruning)

Alternative to the slice pruning at the partition (subgraph) level, the cut pruning at the vertex level is to prune some insignificant vertices with their associated edges by using cut density as a statistical measure to evaluate the significance of a vertex cut. Similar to matrix

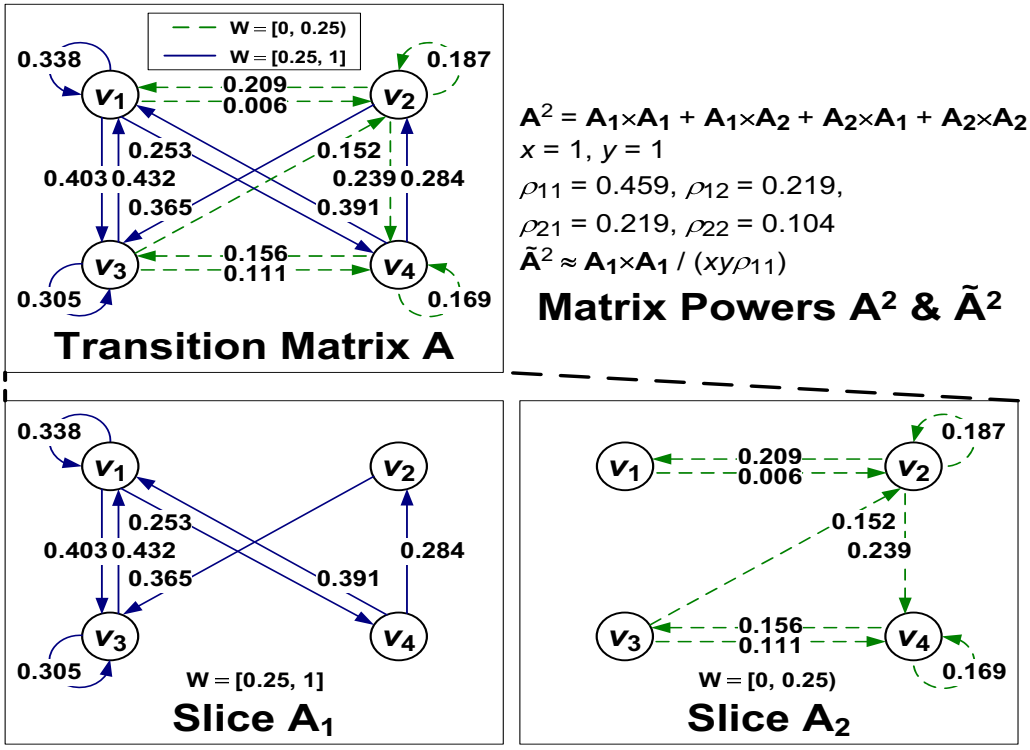


Figure 5.8: Matrix Power with Slice Pruning

multiplication, a fast Monte-Carlo approach is used to further improve the performance of strip multiplications with bounded error.

Definition 20 (CutDensity) *Let K=(S, D_K, E_K, W_J) be a strip of an in-edge slice J=(S, D, E_J, W_J) defined in Eq.(8.4) where $D_K \subseteq D$ and E_K=$\{(u, v)|u \in S, v \in D_K, (u, v).weight \in W_J, (u, v) \in E_J\}$, and $L(u)$=(u, D_K, E_L, W_J) be an out-edge cut of K where $u \in S$ and E_L=$\{(u, v)|v \in D_K, (u, v) \in E_K, (u, v).weight \in W_J\}$, the density of $L(u)$ is defined as follow.*

$$CutDensity(L(u)) = |L(u)| = \sqrt{\sum_{v \in D_K} \sum_{(u,v) \in E_L} w(u, v)^2} \tag{5.5}$$

where $L(u)$ denotes the out-edge cut itself and its (row) vector representation, $|L(u)|$ is the magnitude of vector $L(u)$, and $w(u, v)$ denotes the weight of an edge (u, v). If an out-edge cut has many edges with large weights, then it will have a large density. The density definition of an in-edge cut, corresponding to a column vector, is similar to Eq.(5.5).

Algorithm 7 StripMultiply(A, k, x, B, l, y, z)

1: Initialize $E[1 \cdots \lfloor n/q \rfloor][1 \cdots \lfloor n/q \rfloor]$;
2: **for** $m = 1, \cdots, n$
3: $\rho_m = \frac{CutDensity(A_{kxm}) * CutDensity(B_{lym})}{\sum_{i=1}^{n} CutDensity(A_{kxi}) * CutDensity(B_{lyi})}$;
4: **for** $h = 1, \cdots, z$ independently
5: Pick $a \in \{1, \cdots, n\}$ with Prob$(a = m) = \rho_m, m = 1, \cdots, n$;
6: $\Phi = \Phi \cup \{a\}$;
7: $\tilde{M}$ = **PartitionLoad**$(A, k, x, out, strip, \Phi)$;
8: $\tilde{N}$ = **PartitionLoad**$(B, l, y, in, strip, \Phi)$;
9: R = **ParallelJoin**$(\tilde{M}, \tilde{N}, \tilde{M}.D = \tilde{N}.S)$
10: **for each record** in R
11: $E[a.source][b.destination].AddWeight(a.weight * b.weight/(z * \rho_m))$ where m is the index of $a.destination$ and $b.source$;
12: **return** E;

Given an out-edge strip M (i.e., A_{kx}: the x^{th} strip in slice A_k) with n in-edge cuts $M_1, \cdots, M_n$ (i.e., cuts $A_{kx1}, \cdots, A_{kxn}$), and an in-edge strip N (i.e., strip B_{ly}) with n out-edge cuts $N_1, \cdots, N_n$ (i.e., cuts $B_{ly1}, \cdots, B_{lyn}$), we decompose a strip multiplication O=M×N into n cut multiplications, i.e., $O = \sum_{m=1}^{n} M_m \times N_m$. Similarly, the strip multiplication with cut pruning reduces the exact n cut multiplications to the approximate z ($<< n$) cut multiplications, while maintaining the utility with bounded error. The selection probability ρ_m of each pair of cuts (M_m and N_m) is first calculated in terms of their *CutDensity*. The algorithm independently performs z trials to choose z "significant" cut pairs from the original n cut pairs. Φ maintains the z index entries for chosen cuts of M and N in z trials. We then invoke the PartitionLoad routine to load and filter the out-edge strip M and the in-edge strip N with the chosen cuts. The ParallelJoin routine is executed at the cut level to join in-edges (a) and out-edges (b) of z "significant" vertices. The final edge set E is produced by summarizing the pairwise-cut multiplications with rescaling factor $\frac{1}{z*\rho_m}$ to compensate for the cuts that are not picked. Thus, the multiplication is approximated as $\tilde{O} = \sum_{h=1}^{z} \frac{M_{\Phi(h)} \times N_{\Phi(h)}}{z * \rho_{\Phi(h)}}$.

Similarly, the following theoretical results demonstrate that our cut pruning strategy can achieve a good approximation with bounded error.

Theorem 21 *Given an actual multiplication* $O = \sum_{m=1}^{n} M_m \times N_m$, *a multiplication approxima-*

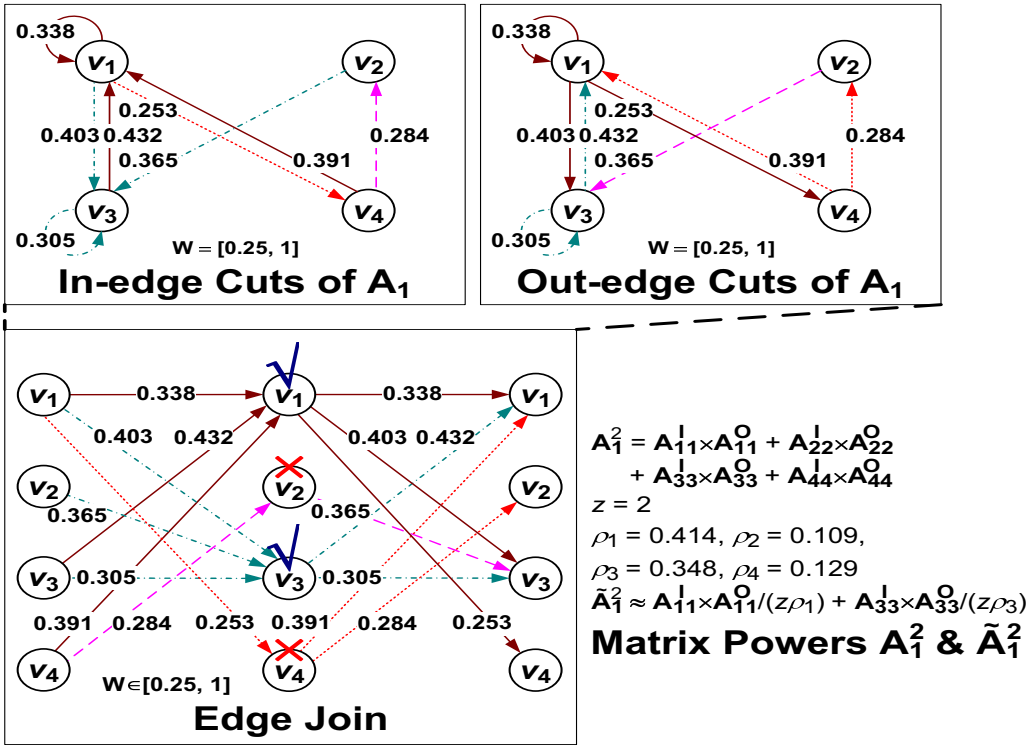


Figure 5.9: Matrix Power with Cut Pruning

tion $\tilde{O}=\sum_{h=1}^{z}\frac{M_{\Phi(h)}\times N_{\Phi(h)}}{z*\rho_{\Phi(h)}}$, *then the expectation* $E(\tilde{O}(i,j))=O(i,j)$ *for* $\forall i, j\in\{1,\cdots,n\}$.

Theorem 22 *Given the same definitions in Theorem 21, then the variance* $Var(\tilde{O}(i,j))=\sum_{m=1}^{n}\frac{M(i,m)^2*N(m,j)^2}{z*\rho_m}-\frac{1}{z}O(i,j)^2$ *for* $\forall i, j\in\{1,\cdots,n\}$.

Theorem 23 *Given the same definitions in Theorem 21,*

$$\min_{\rho_m}\left\{E(\|\tilde{O}-O\|_F^2):\sum_{m=1}^{n}\rho_m=1\right\}=\frac{1}{z}(\sum_{m=1}^{n}|M_m||N_m|)^2-\frac{1}{z}\|O\|_F^2$$
$$when\ \rho_m=\frac{|M_m||N_m|}{\sum_{m=1}^{n}|M_m||N_m|} \tag{5.6}$$

The proof of Theorems 21-23 is omitted due to space limit. The proof methods are similar to that used in Theorems 24-20.

Figure 5.9 shows an example of cut pruning for computing the square of slice $\mathbf{A_1}$ in Figure 5.8. Slice $\mathbf{A_1}$ is organized as 4 in-edge cuts and 4 out-edge cuts respectively. Ochre

Table 5.3: Synthetic Simple Graph Datasets

Graph	#Vertices	#Edges	AvgDeg	MaxIn	MaxOut	DegDist
RMAT	4.3B	17.2B	4	1.8M	737.4K	power-law
ErdosRenyi	1.1B	13.0B	12.1	259.7K	434.8K	Poisson
Random	330M	2.1B	6.5	100K	85.6K	heavy-tailed
Kronecker	88M	1.5B	17.5	365.2K	19.3K	multinomial

Table 5.4: Graph Applications

Application	Graph Type	Core Computation
PageRank [96]	single graph	matrix-vector
SpMV [116]	single graph	matrix-vector
Connected Components [115]	single graph	graph traversal
Matrix Power	two graphs	matrix-matrix
Diffusion Kernel [97]	two graphs	matrix-matrix
AEClass [65]	multigraph	matrix-vector

edges, purple edges, green edges and red edges represent in-edge cuts and out-edge cuts of vertices v_1, v_2, v_3 and v_4, respectively. The exact $\mathbf{A_1}^2$ is equal to the sum of 4 cut multiplications: $\mathbf{A^I_{11}}\times\mathbf{A^O_{11}}$ (ochre edge join), $\mathbf{A^I_{12}}\times\mathbf{A^O_{12}}$ (purple edge join), $\mathbf{A^I_{13}}\times\mathbf{A^O_{13}}$ (green edge join), and $\mathbf{A^I_{14}}\times\mathbf{A^O_{14}}$ (red edge join) where $\mathbf{A^I_{1m}}$ and $\mathbf{A^O_{1m}}$ represent the m^{th} in-edge cut and the m^{th} out-edge cut for vertex v_m respectively. If we want to use only two cut multiplications to approximate $\mathbf{A}_1^2$, then it is highly probable that $\mathbf{A^I_{11}}\times\mathbf{A^O_{11}}$ and $\mathbf{A^I_{13}}\times\mathbf{A^O_{13}}$ are picked with rescaling factors $\frac{1}{z*\rho_1}$ and $\frac{1}{z*\rho_3}$ since ρ_1 and ρ_3 are maximal. The cut pruning stops graph propagation through low degree vertices and edges with small weights.

5.5 Experimental Study

We use several typical iterative graph applications to evaluate the performance of graph processing systems on a set of real-world graphs in Table 5.1 and synthetic graphs in Table 5.3. DBLPS is a single coauthor graph. DBLPM is a coauthor multigraph, where each pair of authors have at most 24 parallel coauthor links, each corresponding to one of 24 research fields, as mentioned in Section 5.3.1. Similarly, we build a friendship multigraph of Last.fm, where each friendship edge is classified into a subset of 21 music genres in terms of the same artists shared by two users. Based on the Recursive Ma-

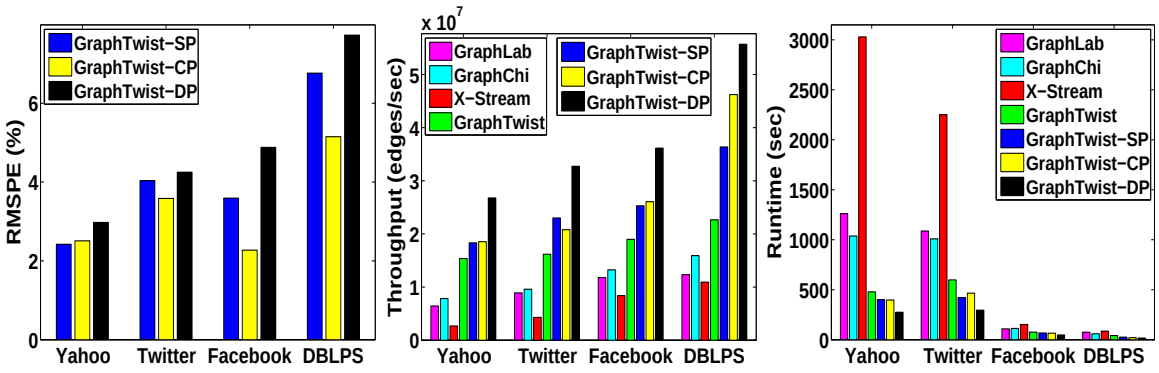
trix (R-MAT) model [117], we utilize the GTgraph suite [118] to generate a scale-free small-world graph with power-law degree distribution. We also use the GTgraph suite [118] based on the Erdos-Renyi random graph model [119] to produce a large-scale random network with Poisson degree distribution. The GenGraph generator [120] is used to construct a large random simple connected graph with heavy-tailed degree distribution. In addition, we employ the SNAP Krongen tool [121] to generate a stochastic Kronecker graph through the Kronecker product. All experiments were performed on a 4-core PC with Intel Core i5-750 CPU at 2.66 GHz, 16 GB memory, and a 1 TB hard drive, running Linux 64-bit. We compare GraphTwist with three existing graph parallel models: **GraphLab** [80], **GraphChi** [81] and **X-Stream** [83]. To evaluate the effectiveness of pruning strategies, we evaluate the following versions of GraphTwist: (1)**GraphTwist** with only access tier abstractions; (2) **GraphTwist-SP** which improves GraphTwist with slice pruning; (3) **GraphTwist-CP** which enhances GraphTwist with cut pruning; and (4) **GraphTwist-DP** which uses both pruning optimizations.

5.5.1 Evaluation Measures

We evaluate the performance of graph processing systems by measuring the running time and the *Throughput* (the number of edges processed per second). We adopt the root-mean-square-percentage-error (RMSPE) between the actual result and the approximate result to evaluate the quality of three versions of GraphTwist with pruning strategies.

$$\begin{aligned} RMSPE(X,\hat{X}) &= \sqrt{\frac{\sum_{i=1}^{n}((x_i-\hat{x}_i)/x_i)^2}{n}} \\ RMSPE(X,\hat{X}) &= \sqrt{\frac{\sum_{i=1}^{n}\sum_{j=1}^{n}((x_{ij}-\hat{x}_{ij})/x_{ij})^2}{n^2}} \end{aligned} \tag{5.7}$$

where x_i is a component in the resulted vector X by the exact computation (matrix-vector computation) and $\hat{x}_i$ is an entry in the approximate vector $\hat{X}$ by the computation with prun-



(a) Quality (b) Throughput (c) Efficiency

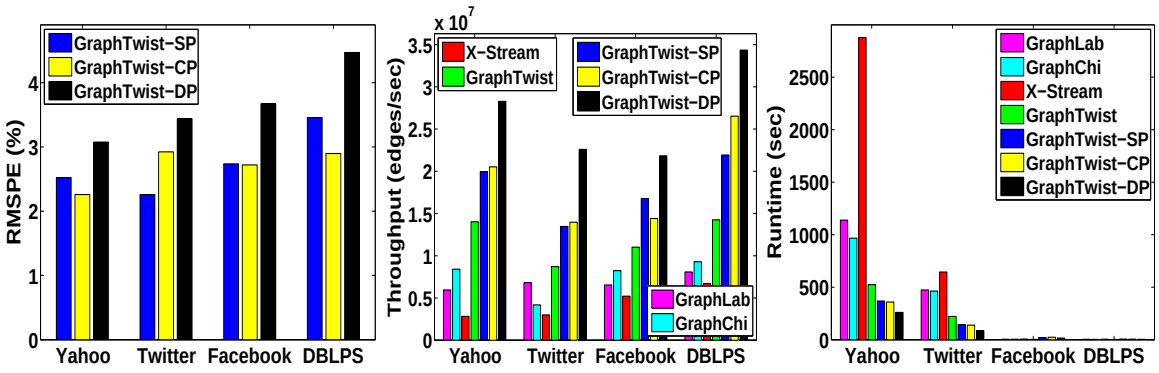
Figure 5.10: PageRank on Four Simple Graphs

ing strategies. Similarly, x_{ij} is an element in the actual result matrix X by matrix-matrix computation and $\hat{x}_{ij}$ is an entry in the approximate result matrix $\hat{X}$. A lower RMSPE number indicates a better performance.

5.5.2 Execution Efficiency on Single Graph

Figures 5.10 and 5.11 present the quality and performance comparison of iterative algorithms on four single graphs by different graph processing systems with different scales: Yahoo, Twitter, Facebook and DBLPS with #*iterations*=1, 5, 40, 30 respectively. Since GraphLab, GraphChi, X-Stream and GraphTwist are the exact graph computations without pruning optimizations, they always have a RMSPE value of zero. Thus, we do not plot the RMSPE bars for four exact graph computations. Figure 5.10 (a) shows the RMSPE values by GraphTwist with different pruning strategies. GraphTwist-DP achieves the highest RMSPE values since it adopts a dual pruning scheme to achieve the highest efficiency. GraphT-wist-CP gains a much lower RMSPE than GraphTwist-SP. The RMSPE values by three approximate systems are smaller than 7.8%, even with #*iterations*=40. This demonstrates that applying pruning techniques to iterative graph applications can achieve a good approximation.

Figure 5.10 (b) exhibits the throughput comparison on four datasets. The throughput values by the exact GraphTwist are larger than 1.53×10^7 and consistently higher than



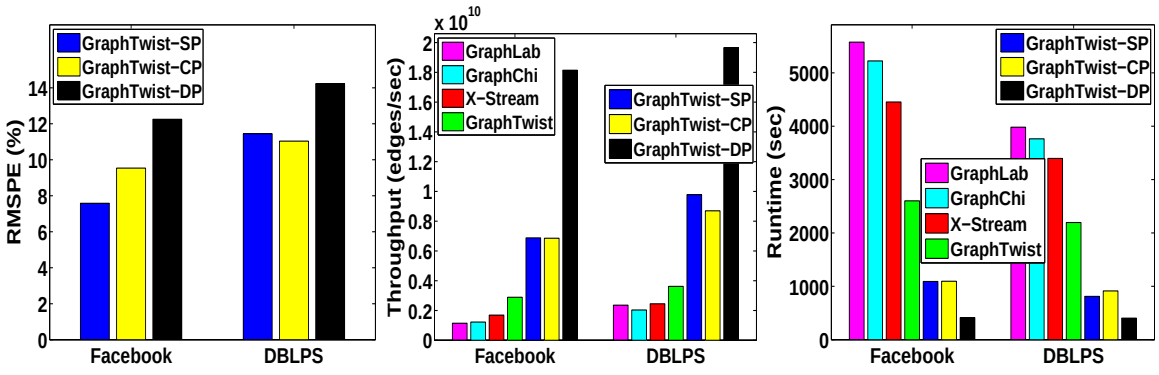
(a) Quality (b) Throughput (c) Efficiency

Figure 5.11: SpMV on Four Simple Graphs

Graph-Lab, GraphChi and X-Stream. All versions of GraphTwist with pruning turned on significantly outperform the exact GraphTwist, with the throughput by GraphTwist-DP as the highest (>2.67×10^7). GraphTwist-SP and GraphTwist-CP achieve slightly lower throughput than GraphTwist-DP.

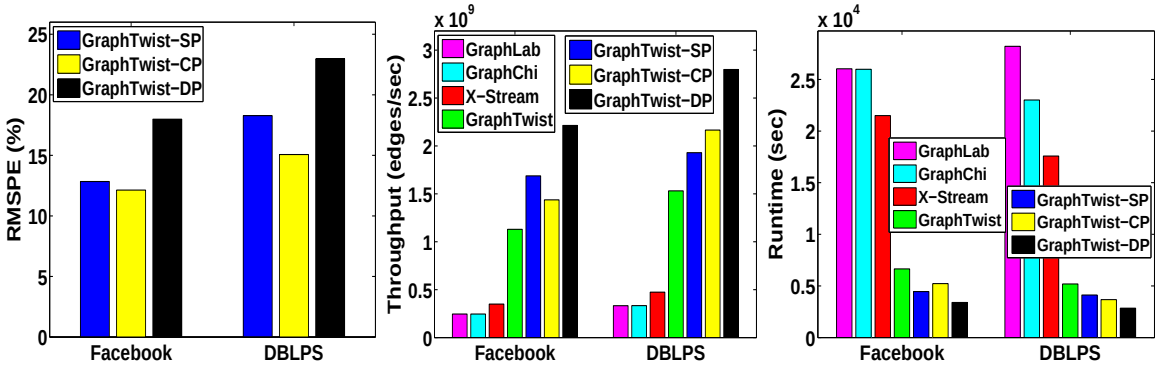
Figure 5.10 (c) compares the running time by different graph parallel models, from loading graph from disk to writing results back to disk. The runtime comparison is consistent with the throughput evaluation in Figure 5.10 (b). We make two interesting observations. First, the exact GraphTwist outperforms GraphLab, GraphChi and X-Stream in all experiments. Second, GraphTwist with different pruning strategies significantly outperform the exact GraphTwist in terms of throughput and runtime while maintaining very good quality in terms of RMSPE.

Similar trends are observed for the performance comparison of SpMV in Figure 5.11. Compared to GraphLab, GraphChi and X-Stream, the exact GraphTwist consistently performs better in all throughput and efficiency tests. All versions of GraphTwist with pruning significantly outperform the exact GraphTwist thanks to the pruning optimizations while maintaining a good approximation.



(a) Quality (b) Throughput (c) Efficiency

Figure 5.12: Matrix Power on Two Simple Graphs

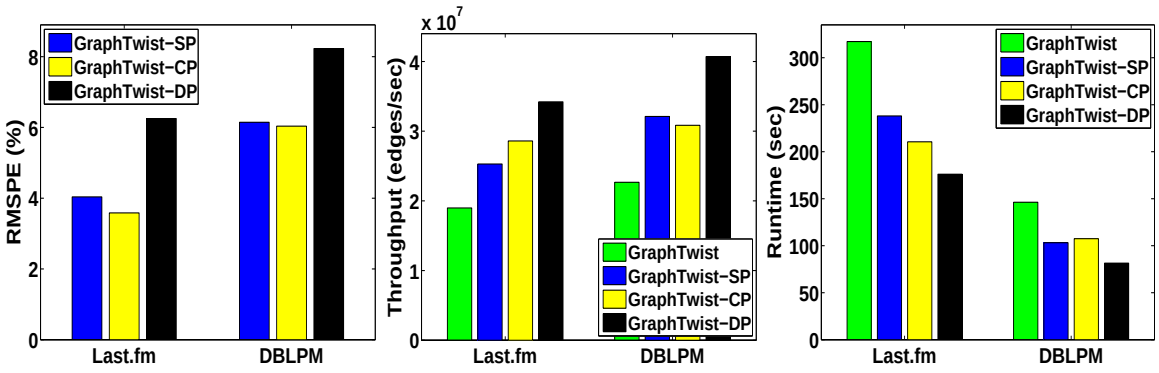


(a) Quality (b) Throughput (c) Efficiency

Figure 5.13: Diffusion Kernel on Two Simple Graphs

5.5.3 Execution Efficiency on Multiple Graphs

Figures 5.12 and 5.13 show the performance comparison of iterative applications on multiple graphs with different graph parallel models. Since GraphLab, GraphChi and X-Stream can not directly address matrix-matrix multiplications among multiple graphs, we thus modify the corresponding implementations to run the above graph applications. As the complexity of matrix-matrix multiplication ($O(n^3)$) is much larger than the complexity of matrix-vector multiplication ($O(n^2)$), we only compare the performance by different graph processing systems on two smaller datasets: Facebook and DBLPS. We observe the very similar trends as those shown in Figures 5.10 and 5.11. All versions of GraphTwist significantly prevail over GraphLab, GraphChi and X-Stream in all efficiency tests, and

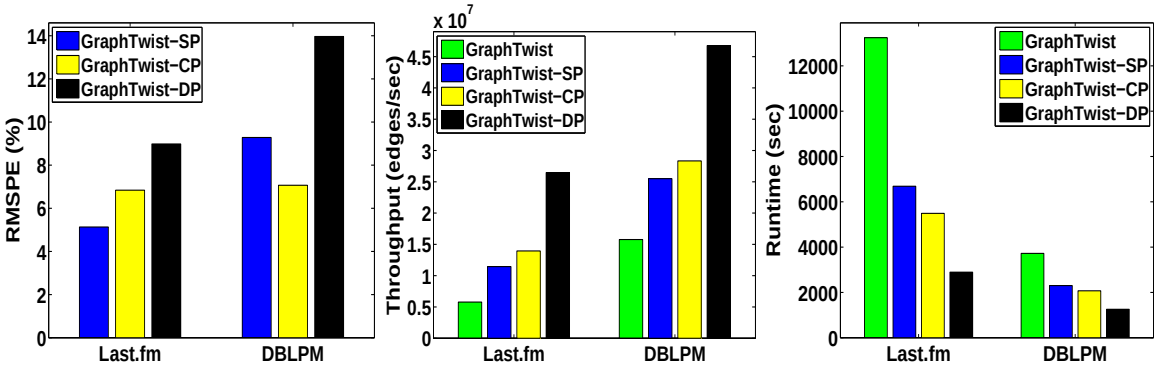


(a) Quality (b) Throughput (c) Efficiency

Figure 5.14: PageRank on Multigraph
GraphTwist-DP with double pruning strategies obtains the highest throughput.

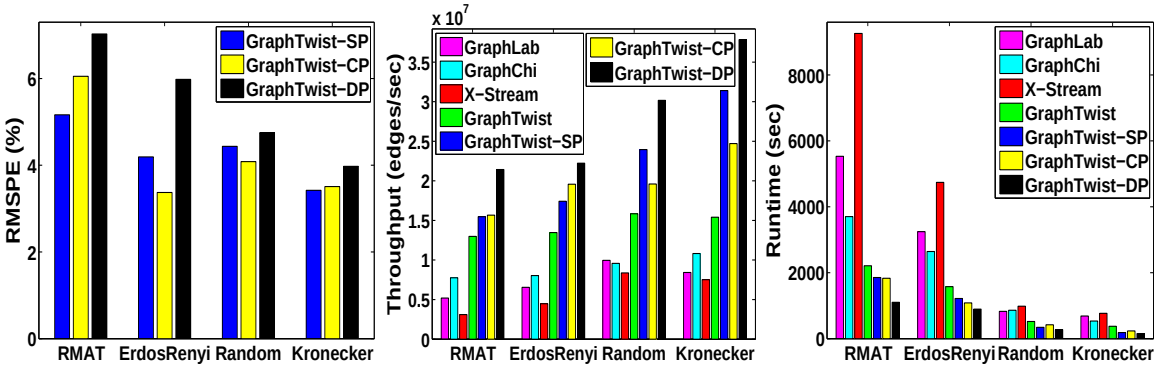
5.5.4 Execution Efficiency on Multigraph

Since existing representative graph processing systems can not address iterative applications on multigraphs, we only perform the efficiency comparison of PageRank on multigraphs by different GraphTwist versions, as shown in Figure 5.14. GraphTwist partitions the 3D cube of a multigraph into p slices along dimension W. Each slice consists of parallel edges with a unique semantics, say the DBLP coauthored papers in the area of DB and the Last.fm user friendships with respect to pop music genre. By hashing the parallel edges with the same semantics into the same partition, each slice corresponds to one partition, and represents a subgraph with only those edges that have the corresponding semantics included in the hash bucket for that partition. The PageRank algorithm is executed on each slice with a unique semantics in parallel to compute the ranking vector of authors (or users) in each research field (or music genre). Figure 5.15 presents the performance comparison of a multigraph algorithm (AEClass) with the GraphTwist implementation. AEClass [65] transforms the problem of multi-label classification of heterogeneous networks into the task of multi-label classification of coauthor (or friendship) multigraph based on activity-based edge classification. We observe very similar trends as those shown in Figures 5.10-5.13. All three GraphTwist versions with pruning achieve relatively lower RMSPE (<14.0%).



(a) Quality (b) Throughput (c) Efficiency

Figure 5.15: AEClass on Multigraph



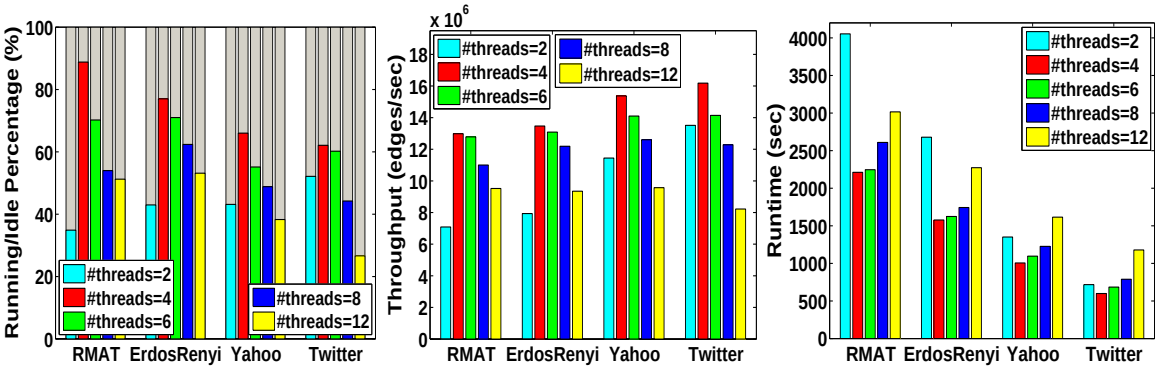
(a) Quality (b) Throughput (c) Efficiency

Figure 5.16: PageRank on Four Synthetic Simple Graphs

GraphTwist-DP obtains the highest throughput (>2.6×10^7), while the throughput by the exact GraphTwist is the lowest (>5.7×10^6). The throughput values by GraphTwist-SP and GraphTwist-CP stand in between (>1.1×10^7).

5.5.5 Execution Efficiency on Synthetic Graphs

Figure 5.16 shows the performance comparison of GraphTwist with other three systems by PageRank on four synthetic graphs with *#iterations*=2, 2, 5, 5 respectively. We observe similar trends to the performance on real-world graphs. GraphTwist consistently outperforms GraphLab, GraphChi and X-Stream in both throughput and efficiency tests. GraphTwist with all versions of pruning significantly outperform the exact GraphTwist thanks to the pruning optimizations while maintaining a good approximation.

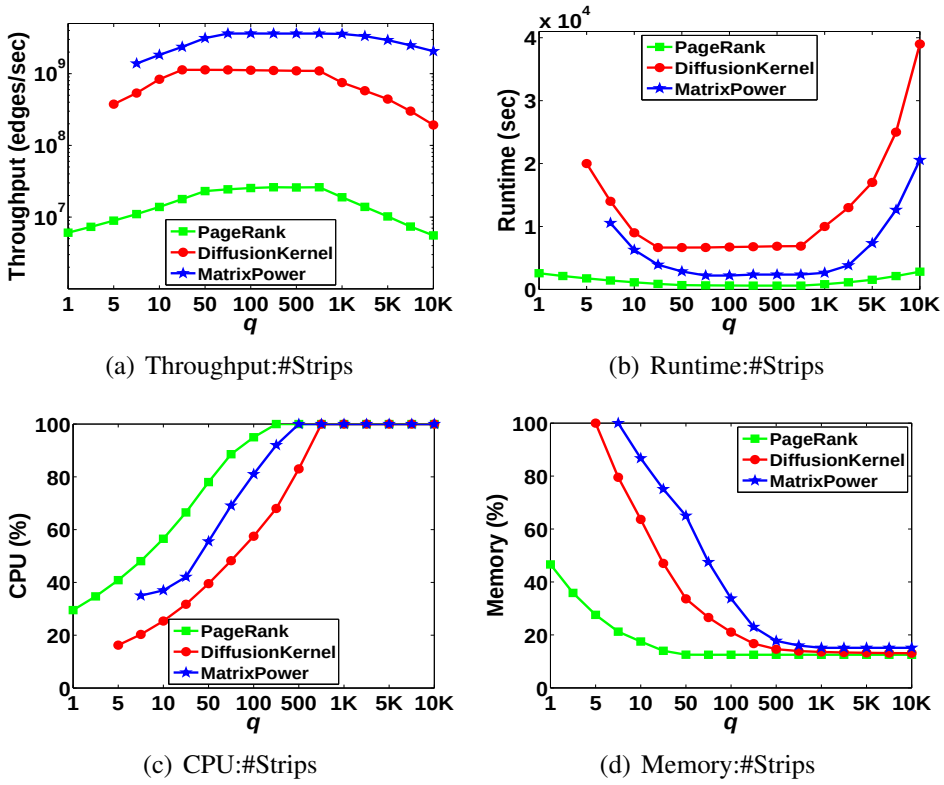


(a) Utilization (b) Throughput (c) Efficiency

Figure 5.17: PageRank by GraphTwist wrt Varying Threads

5.5.6 Impact of #Threads

Figure 5.17 presents the performance comparison of GraphTwist by running PageRank on two synthetic simple graphs and two real simple graphs with #*iterations*=2, 2, 3, 5 respectively. Figure 5.17 (a) shows the average utilization comparison (computation/non-computation runtime percentage) of each thread by varying #*threads* from 2, 4, 6, 8, to 12. Recall Section 5.3.2, two levels of threads are used in GraphTwist: partition-level threads to process multiple partitions (slices, dices or strips) in parallel. Within each partition thread, we also execute multiple threads at vertex level. The total number of threads is arranged by $h*l$ (h is #*threads* at partition level and l is #*threads* at vertex level). For example, for the setup of 8 threads, we use 2 threads at partition level and 4 threads at vertex level. We compare the total runtime to execute a task of iterative graph computation, including thread running time and thread idle runtime. The thread idle runtime is measured when performing non-computation operations, including thread waiting, disk I/O, context switching and CPU scheduling. Figure 5.17 (a) shows that the utilization rate on all four graphs with 4, 6 or 8 threads are better compared to 2 or 12 threads. When #*threads*=2, the threads are busy at <57% of time and when #*threads*=12, the threads are idle at >46% of time. In both cases, the threads do not efficiently utilize CPU resource. Figure 5.17 (c) (or Figure 5.17 (b)) measure the performance impact of parallel threads by PageRank on four simple graphs. We have observed that the runtime is very long when the number of parallel threads is



(a) Throughput:#Strips (b) Runtime:#Strips

(c) CPU:#Strips (d) Memory:#Strips

Figure 5.18: Impact of #Strips

relatively small (#*threads*=2) or very large (#*threads*=12) and it is almost a stable horizontal line when #*threads*=4, 6, 8). Also GraphTwist usually achieves the best performance by spawning between #*cores* and 2∗#*cores* threads because less #*threads* (<#*cores*) often lead to underutilization of available CPU resource in graph parallel system. On the other hand, more threads (>2∗#*cores*) may introduce additional non-computation overhead, such as context switching and CPU scheduling, and thus hurt system performance.

5.5.7 Decision of #Partitions

Figure 5.18 measures the performance impact of different numbers of strips on GraphTwist with PageRank over Twitter, Diffusion Kernel on Facebook, and Matrix Power on DBLPS. The x-axis shows different settings of the number of strips. We vary #*Strips* from 1 to 10,000 and fix #*Slices* and #*Dices* as 5 in each figure. It is observed that the runtime curve (or the throughput curve) for each application in each figure follows a similar "U" curve (inverted "U" curve) with respect to the size of strip, i.e., the runtime is very long

when the unit size is relatively small or very large and it is almost a stable horizontal line when the unit size stands in between two borderlines. This is because bigger strips often lead to substantial work imbalance in graph applications. On the other hand, smaller strips may result in frequent external storage access and lots of page replacements between units lying in different pages. Figure 5.18 (c) measures the CPU utilization by GraphTwist for three real applications. The CPU utilization rate of each application increases quickly when $\#Strips$ is increasing. This is because for the same graph, the larger number of strips gives the smaller size per strip and the smaller strips in big graphs often lead to better workload balancing for parallel computations. Figure 5.18 (d) shows the memory utilization comparison. The memory curves are totally contrary to the corresponding CPU curves: the smaller the number of strips, the larger size each strip will have, thus the larger the memory usage. The performance impact of $\#Slices$ or $\#Dices$ on GraphTwist have similar trends to Figure 5.18.

5.6 Conclusion

We present a scalable, efficient, provably correct two-tier graph parallel processing system, GraphTwist. At storage and access tier, GraphTwist employs three customizable parallel abstractions: slice partitioning, strip partitioning and dice partitioning, to maximize parallel computation efficiency. At computation tier, GraphTwist presents slice pruning and cut pruning strategies. Our pruning methods are utility-aware and can significantly speed up the computational performance while preserving the computational utility defined by the graph applications.

CHAPTER 6

SERVICERANK: RANKING SERVICES BY SERVICE NETWORK STRUCTURE AND SERVICE ATTRIBUTES

Services related to one another in terms of providers, consumers and context often form a service network with vertices representing services and edges representing connectivity relationships between services. Service network analysis is an essential aspect of web service discovery, search, mining and recommendation. Many popular web service networks are content-rich in terms of heterogeneous types of entities, attributes and links. A main challenge for ranking services is how to incorporate multiple complex and heterogeneous factors, such as service attributes, relationships between services, relationships between services and service providers or service consumers, into the design of service ranking functions. In this chapter, we model services, attributes, and the associated entities, such as providers, consumers, by a heterogeneous service network. We present a unified neighborhood random walk distance measure, which integrates various types of links and vertex attributes by a local optimal weight assignment. Based on this unified distance measure, a reinforcement algorithm, ServiceRank, is provided to tightly integrate ranking and clustering by mutually and simultaneously enhancing each other such that the performance of both can be improved. For example, ServiceRank ensures that highly ranked services in a cluster should be provided by many highly ranked providers in the same cluster. Highly ranked providers in a cluster should provide many highly ranked services in the same cluster. To further improve the quality of the clustering and ranking, an additional clustering matching strategy is proposed to efficiently align clusters from different types of objects. Our extensive evaluation on both synthetic and real service networks demonstrates the effectiveness of ServiceRank in terms of the quality of both clustering and ranking among multiple types of entity, link and attribute similarities in a service network.

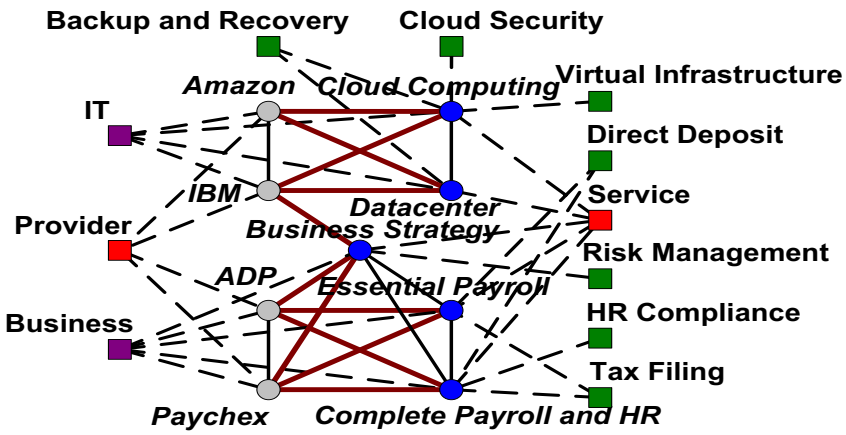


Figure 6.1: A Heterogeneous Service Network from IBM Knowledge Base

6.1 Introduction

With the increasing popularity of web services, web service management is becoming an interesting and challenging research problem which has received much attention recently [122, 123, 124, 125, 126, 127, 128, 129]. Service network analysis has emerged as a critical aspect of web service management in both industry and academic research. Many popular web service networks are content-rich in terms of heterogeneous types of entities and links, associated with incomplete attributes. Such web service networks expose two heterogeneity challenges: (1) Multiple types of entities co-exist in the same service network with various attributes, and (2) Links between entities have different types and carry different semantics. Figure 8.1 presents a real service network from IBM knowledge base. There are two kinds of object vertices: blue service vertices and grey provider nodes. Each service vertex may contain three types of properties: red, purple and green attribute vertices specify service's "Type", "Category" and "Capability", respectively. Each provider vertex may have two kinds of attributes: red "Type" attribute and purple "Category" property. On the other hand, there are three kinds of links: an ochre edge represents the "Provides" relationship between services and providers; a black line specifies the structure relationship between objects with the same type; a dashed edge denotes the attribute edge between object and its attribute.

Typical applications of web service management include service retrieval and selection, service ranking and recommendation, service clustering and discovery, service adaptation and composition etc. Among these applications, service clustering and ranking are two significant ones. Ranking entails assigning a score to each service, quantifying its characteristics based on some ranking criteria. With such criteria, "interesting" services to response a specific user request appear high in the returned list. Prominent web ranking algorithms such as HITS [130] and PageRank [96] as well as service ranking methods such as [124] and [125] rank webpages or services on the whole homogeneous network so that the most relevant webpages or services are presented on the top of the returned list. On the other hand, clustering partitions a service network into groups so that services within a cluster are densely connected based on a certain similarity measure. Most existing graph clustering techniques have focused on the topological structures of homogeneous graph based on various criteria.

Although clustering and ranking are two well-known analytical tools for web service management, existing service clustering and ranking approaches are usually regarded as two independent processes. As a result, two main drawbacks characterize such approaches. First, clustering the entire service network without considering service's ranking may result in incorrect clustering result. Suppose we remove providers "Amazon", "Paychex" and their associated links from Figure 8.1, then service "Business Strategy" has only two providers "IBM" and "ADP". If we want to partition vertices into two clusters, then "IBM" and "ADP" will be in cluster "IT" and cluster "Business", respectively. But there is no statistically convincing evidence on which cluster "Business Strategy" should fall into. Second, ranking services on the whole service network without considering which groups they belong to often leads to biased ranking result. If we adopt the global ranking algorithm to rank all services in Figure 8.1, then service "Business Strategy" has a higher score than service "Datacenter" since "Business Strategy" has more peer services and is provided by more providers. This ranking result does not make sense for a customer

seeking a "IT" service. However, integrating clustering and ranking together can lead to more comprehensible results. By combining clustering and ranking, "Business Strategy" and "ADP" will belong to cluster "Business" since both rank high in cluster "Business" and rank low in cluster "IT". In contrast, "IBM" ranks high in cluster "IT" and relatively low in "Business". Thus, "Business Strategy" and "ADP" are more similar. Similarly, service "Datacenter" will have a higher score than service "Business Strategy" in cluster "IT" if we rank all services in each cluster.

In this chapter, we identify four requirements for ranking and clustering a heterogeneous network of services. First, we need to integrate various types of links and attributes into a unified distance model to estimate the pairwise vertex closeness. Second, we need to design a robust probabilistic clustering method for heterogeneous service network to make our approach applicable to a wide range of applications. Third, we need to design a more useful service ranking approach with combining ranking information from peer services, providers and associated attributes. Finally, we need to develop a framework that can smoothly integrate ranking and clustering techniques.

The main contributions of this chapter are outlined below. First, we propose a unified neighborhood random walk distance measure integrating various types of link and attribute information with the local optimal weight assignment on a heterogeneous service network. Second, we propose a greedy strategy to efficiently execute clustering matching process to align clusters for each type of objects on the heterogeneous service network. Third, we present a general algorithm that smoothly integrates ranking and clustering techniques, while mutually enhances the individual performance of each of those techniques. Finally, we perform extensive evaluation of our proposed ranking approach on both synthetic and real service datasets to demonstrate the effectiveness and efficiency of our method.

6.2 Related Work

Web service discovery and management has been a heated topic in recent years [122, 126, 127, 129]. Skoutas et al. [124] proposed a methodology for ranking and clustering the relevant web services based on the notion of dominance, which apply multiple matching criteria without aggregating the match scores of individual service parameters. Xiao et al. [123] proposed a context modeling approach which can dynamically handle various context types and values. Based on the relations among context values, the algorithm can capture the potential services that the user might need. Almulla et al. [125] presented a web services selection model based on fuzzy logic and proposed a fuzzy ranking algorithm based on the dependencies between proposed quality attributes. Liu et al. [128] proposed a heuristic social context-Aware trust network discovery algorithm, H-SCAN, by adopting the K-Best-First Search (KBFS) method and some optimization strategies.

Graph ranking is one of the core tasks in social networks. Most of existing graph ranking techniques [130, 96, 131, 132, 133] compute ranking scores by only resorting to graph structure information. Jeh and Widom [134] designed a measure called SimRank, which defines the similarity between two vertices in a graph by their neighborhood similarity. DivRank [135], based on a reinforced random walk in an information network, can automatically balances the prestige and the diversity of the top ranked vertices in a principled way. Tong et al. [136] defined a goodness measure to capture both the relevance and the diversity for a given ranking list.

Graph clustering has attracted active research in the last decade. Most of existing graph clustering techniques have focused on the topological structures based on various criteria, including normalized cuts [11], modularity [12], structural density [13]. Some recent works, SA-Cluster [20] and BAGC [21], perform clustering based on both structural and attribute similarities to partition the collaboration graph with single link type and single attribute type into k clusters. Sun et al. [23] proposed GenClus to cluster general heteroge-

neous information networks with different link types and different attribute types.

6.3 Problem Statement

In this section, we first introduce the problem formulation of service clustering and ranking considering both service network structure and service attribute information.

A **heterogeneous service network** is denoted as $G = (V, A, E)$, where V represents the set of object vertices, A denotes the set of property vertices, E contains multiple types of edges between object or property vertices. V contains two types of object vertices: the service set S and the provider set P, i.e., $V = S \cup P$. $A = \{a_{11}, \ldots, a_{1n_1}, \ldots, a_{m1}, \ldots, a_{mn_m}\}$ represents m associated attributes and their values for describing object properties. $Dom(a_i) = \{a_{i1}, \ldots, a_{in_i}\}$ represents the domain of attribute a_i with a size of $|Dom(a_i)| = n_i$. An attribute vertex $v_{ij} \in A$ represents that attribute a_i takes the j^{th} value. An attribute edge $(v_i, v_{jk}) \in E$ iff vertex v_i takes the value of a_{jk} on attribute a_j. Thus, there are five types of edges between different kinds of vertices: $E = E_{SS} \cup E_{PP} \cup E_{SP} \cup E_{SA} \cup E_{PA}$ where subscripts S, P and A represent the service set, the provider set and the attribute set, respectively. For ease of presentation, we call an edge between object vertices, i.e., $e \in E_{SS} \cup E_{PP} \cup E_{SP}$, as a structure edge and an edge between object and attribute, i.e., $e \in E_{SA} \cup E_{PA}$, as an attribute edge.

In such a heterogeneous service network with various types of objects, links and attributes, a good measure of "similarity" between objects is crucial to efficient social network analysis on real applications. A **unified neighborhood random walk distance measure** is to measure the closeness between vertices based on connectivity, vicinity and transition probabilities at different types of vertices.

Given k initial disjoint clusters of $G = (V, A, E)$ for each type of objects, $S = \cup_{i=1}^{k} S_i$ ($S_i \cap S_j = \emptyset, \forall i \neq j$) and $P = \cup_{i=1}^{k} P_i$ ($P_i \cap P_j = \emptyset, \forall i \neq j$) as the initial service influence, **service influence based probabilistic clustering** is to execute service influence propagation to partition each service $s \in S$ into each of k clusters based on the dynamic social activities

with the initial disjoint service clusters and provider clusters. In the final clustering result, $\sum_{i=1}^{k} p_{si} = 1, \forall s \in S$ where p_{si} represents the normalized probability of service s will be partitioned into the i^{th} cluster. A desired clustering of a heterogeneous service network should achieve a good balance between the following two properties: (1) services within one cluster are close to each other in terms of structure links, while services between clusters are distant from each other; and (2) services within one cluster have similar properties, while services between clusters could have quite different attribute values.

Given a service influence based probabilistic clustering, **service influence based ranking** is to rank each service $s \in S$ and each provider $p \in P$ in each of k clusters based on some heuristics rules so that good service ranking generates good provider ranking, good provider ranking promotes good service ranking.

6.4 A Unified Weighted Distance Measure

In a heterogeneous service network, each service is associated with a service set and a provider set through structure links and an attribute set through attribute links, we propose to use a unified distance measure based on the neighborhood random walk model to integrate various types of structural and attribute similarities. In the heterogeneous service network, there exists a random walk path between two services $s_1, s_2 \in S$ if (1) s_1 and s_2 have the same peer service $s_3 \in S$; (2) both s_1 and s_2 are provided by the same provider $p \in P$; or (3) s_1 and s_2 have the same attribute value $a \in A$. If there are multiple paths connecting s_1 and s_2, then they are close. On the other hand, if there are very few or no paths between s_1 and s_2, then they are far apart.

Definition 21 (Transition Probability Matrix) *Let S be the service set, P be the provider set, and A be the set of associated attribute vertices, the transition probability matrix T of*

a heterogeneous service network G is defined as follow.

$$T = \begin{bmatrix} T_{SS} & T_{SP} & T_{SA} \\ T_{PS} & T_{PP} & T_{PA} \\ T_{AS} & T_{AP} & T_{AA} \end{bmatrix} \tag{6.1}$$

where T_{SS} is a $|S|\times|S|$ matrix representing the transition probabilities between service vertices; a $|P|\times|P|$ matrix T_{PP} specifies the transition probabilities between provider vertices; T_{SP} or T_{PS} represents the transition probabilities between services and providers; T_{SA} or T_{AS} denotes the transition probabilities between services and attributes; T_{PA} or T_{AP} represents the transition probabilities between providers and attributes; and T_{AA} is a $|A|\times|A|$ matrix with all 0s since there is no edge between attribute vertices. Here, $|S|$, $|P|$ and $|A|$ represent the cardinalities of the service set S, the provider set P and the attribute set A, respectively.

Since each type of structure and attribute edges may have different degrees of contribution in random walk distance, we assign each type of edges an individual weight. T_{SS}, T_{SP}, T_{PS} and T_{PP} correspond to four kinds of structure edges, and the corresponding structure weights are defined as α_{SS}, α_{SP}, α_{PS} and α_{PP}, respectively. Notice that α_{SP} is equal to α_{PS} since edges have no orientation in an undirected service network. On the other hand, there are m associated attributes with object vertices in the service network. The attribute edges connected to attribute vertices $v_{i1},\ldots,v_{in_i}$ corresponding to attribute a_i are assigned to an attribute weight β_i. We proposed a dynamic weight tuning method [137] to produce a local optimal weight assignment for various types of links. Based on this weight assignment, each submatrix in T is defined as follow.

$$T_{SS}(i,j) = \begin{cases} \alpha_{SS} e_{ij}, & if (v_i, v_j) \in E_{SS} \\ 0, & otherwise \end{cases} \tag{6.2}$$

where e_{ij} denotes the number of providers that service i and service j co-shared. When

service i and service j have different types or are from different categories, e_{ij} is usually equal to 0 *since they often do not have the same providers.*

$$T_{PP}(i,j) = \begin{cases} \alpha_{PP} e_{ij}, \ if(v_i, v_j) \in E_{PP} \\ 0, \qquad otherwise \end{cases} \tag{6.3}$$

where e_{ij} is the number of services that provider i and provider j co-provided.

$$T_{PS}(i,j) = \begin{cases} \alpha_{PS} e_{ij}, \ if(v_i, v_j) \in E_{PS} \\ 0, \qquad otherwise \end{cases} \tag{6.4}$$

where e_{ij} is the number of service j that provider i provided. e_{ij} has a value of 0 *or* 1 *since provider i may or may not provide service j in the original dataset. As the relationship between services and providers is symmetric, $T_{PS}(i,j)$ is equal to $T_{SP}(j,i)$ due to $\alpha_{SP} = \alpha_{PS}$.*

$$T_{SA}(i,J) = \begin{cases} \beta_j e_{ijk}, \ if(v_i, v_{jk}) \in E_{SA} \\ 0, \qquad otherwise \end{cases}, \ J = k + \Sigma_{l=1}^{j-1} n_l \tag{6.5}$$

where e_{ijk} denotes whether service i takes the k^{th} value on attribute a_j. It also has a value of 0 *or* 1.

$$T_{AS}(I,j) = \begin{cases} e_{ijk}, \ if(v_j, v_{ik}) \in E_{SA} \\ 0, \qquad otherwise \end{cases}, \ I = k + \Sigma_{l=1}^{i-1} n_l \tag{6.6}$$

where e_{ijk} specifies whether the k^{th} value on attribute a_i is taken by service j. The weight factor is ignored since each row in T_{AS} corresponds to the same attribute vertex v_{ik}.

Since both services and providers are the subclass of objects, T_{PA} and T_{SA} have the similar definitions and so have T_{AP} and T_{AS}.

$$T_{PA}(i,J) = \begin{cases} \beta_j e_{ijk}, \ if(v_i, v_{jk}) \in E_{PA} \\ 0, \qquad otherwise \end{cases}, \ J = k + \Sigma_{l=1}^{j-1} n_l \tag{6.7}$$

where e_{ijk} denotes whether provider i takes the k^{th} value on attribute a_j.

$$T_{AP}(I, j) = \begin{cases} e_{ijk}, \ if(v_j, v_{ik}) \in E_{PA} \\ 0, \qquad otherwise \end{cases}, \ I = k + \Sigma_{l=1}^{i-1} n_l \tag{6.8}$$

where e_{ijk} specifies whether the k^{th} value on attribute a_i is taken by provider j.

Since each row of the transition probability matrix should sum to 1*, we then perform the row-wise normalization for T. Entries $T_{SS}(i, j)$, $T_{SP}(i, j)$ and $T_{SA}(i, j)$ are normalized by dividing them by the sum of row entries, i.e., $\Sigma_{l=1}^{|S|} T_{SS}(i, l) + \Sigma_{l=1}^{|P|} T_{SP}(i, l) + \Sigma_{l=1}^{|A|} T_{SA}(i, l)$. Similarly, elements $T_{PS}(i, j)$, $T_{PP}(i, j)$ and $T_{PA}(i, j)$ are normalized by $\Sigma_{l=1}^{|S|} T_{PS}(i, l) + \Sigma_{l=1}^{|P|} T_{PP}(i, l) + \Sigma_{l=1}^{|A|} T_{PA}(i, l)$. Entries $T_{AS}(i, j)$ and $T_{AP}(i, j)$ are normalized by $\Sigma_{l=1}^{|S|} T_{AS}(i, l) + \Sigma_{l=1}^{|P|} T_{AP}(i, l)$ since T_{AA} is a zero matrix.*

A random walk on a heterogeneous service network G is performed in the following way. Suppose a particle starts at a certain vertex v_0 and walks to a vertex v_s in the s^{th} step and it is about to move to one of the neighbors of v_s, denoted as $v_t \in N(v_s)$, with the transition probability $T(s, t)$, where $N(v_s)$ contains all neighbors of vertex v_s. The vertex sequence of the random walk is a Markov chain. The probability of going from v_i to v_j through a random walk of length l can be obtained by multiplying the transition probability matrix l times.

Definition 22 (Unified Neighborhood Random Walk Distance) *Let T be the transition probability of a heterogeneous service network G, l be the length that a random walk can go, and $c \in (0, 1)$ be the restart probability, the unified neighborhood random walk distance $d(u, v)$ from vertex u to vertex v in G is defined as follow.*

$$d(u, v) = \sum_{\substack{\tau: u \rightsquigarrow v \\ length(\tau) \leq l}} p(\tau) c (1 - c)^{length(\tau)} \tag{6.9}$$

where τ is a path from u to v whose length is length(τ) with transition probability $p(\tau)$.

$d(u, v)$ reflects the vertex closeness based on multiple types of links among services, providers and attributes.

The matrix form of the unified distance is given as follow.

$$R = \sum_{\gamma=1}^{l} c(1-c)^{\gamma} T^{\gamma} \tag{6.10}$$

For ease of presentation, R is rewritten as the following form by using the similar representation in Eq.(8.1).

$$R = \begin{bmatrix} R_{SS} & R_{SP} & R_{SA} \\ R_{PS} & R_{PP} & R_{PA} \\ R_{AS} & R_{AP} & R_{AA} \end{bmatrix} \tag{6.11}$$

6.5 Service Influence Based Clustering

In this section, we propose an innovative service influence based probabilistic clustering framework considering social interactions among service and provider disjoint clusters.

6.5.1 Clustering Matching Process

Most existing clustering methods such as k-Means [138] and k-Medoids [25] only offer disjoint clusters. We can not directly apply them to our heterogeneous service network, otherwise they will lead to two issues: (1) service clusters and provider clusters are independent of each other; and (2) each object is partitioned into a single cluster. To address the first issue, the clustering matching process is designed to produce a one-to-one matching between service clusters and provider clusters based on some similarity measures.

The inter-cluster similarity with the same type is defined as follow.

$$d(\overline{c}_L) = \frac{1}{k(k-1)} \sum_{i=1}^{k} \sum_{j=1, j\neq i}^{k} d(c_L^i, c_L^j),\ L \in \{S, P\} \tag{6.12}$$

where c_L^x represents the centroid of cluster L_x and L specifies the service set (or the provider set). $d(c_L^i, c_L^j)$ is the unified distance from c_L^i to c_L^j. $S = \cup_{i=1}^{k} S_i$ $(S_i \cap S_j = \emptyset, \forall i \neq j)$ (or $P = \cup_{i=1}^{k} P_i$ $(P_i \cap P_j = \emptyset, \forall i \neq j)$) is a disjoint clustering of services (or providers). A clustering of services (or providers) with a small inter-cluster similarity is considered as a good clustering based on this criterion.

The inter-cluster similarity across different types is designed to quantitatively measure the similar extent between a service cluster and a provider cluster. It is defined as follow.

$$d(X_i, Y_j) = \frac{1}{|X_i||Y_j|} \sum_{x \in X_i, y \in Y_j} d(x, y) \; (X, Y \in \{S, P\}, X \neq Y) \tag{6.13}$$

where X is a disjoint clustering of services (or providers) and Y is a disjoint clustering of providers (or services).

The disjoint clustering matching algorithm is presented in Algorithm 8. It first chooses the clustering with the minimal inter-cluster similarity as the basic clustering X and the clustering with the maximal inter-cluster similarity as the clustering Y to be matched. The algorithm then calculates the inter-cluster similarity between X's clusters and Y's clusters and matches each cluster of Y to the cluster of X with the maximal inter-cluster similarity. It places the potential conflict cluster labels of Y and the corresponding cluster label of X into Q. During each "Dequeue" iteration, the algorithm rematches the one of two conflict clusters Y_m and Y_n, which has smaller inter-cluster similarity with cluster X_i, and puts new possible conflict tripes into Q until the queue is empty. Finally, it updates the cluster labels of Y with the matched cluster labels respectively.

6.5.2 Service Influence Propagation

The process of service influencing one another in a service network is very similar to the heat diffusion process [3, 1]. Heat diffusion is a physical phenomenon that heat always flows from an object with high temperature to an object with low temperature. In the context of a heterogeneous service network, an early service often influences other late ser-

Algorithm 8 Disjoint Clustering Matching

Input: a heterogeneous service network $G = (V, A, E)$, a disjoint clustering $S_1, \ldots, S_k, P_1, \ldots, P_k$, a unified random walk distance matrix R, and a queue Q containing triples of conflict cluster labels.

Output: a $1 \times k$ vector CL containing new cluster label for the matched clusters of S or P.

$X = argmin_{L \in \{S,P\}} d(\overline{c}_L)$; $Y = argmax_{L \in \{S,P\}} d(\overline{c}_L)$;
$CL = (0, 0, \cdots, 0)$; $Q = \phi$;
Calculate $d(X_i, Y_j)$ for $\forall i, j = 1, \cdots, k$;
for $j = 1, \cdots, k$
  $CL(j) = argmax_{i \in \{1, \cdots, k\}} d(X_i, Y_j)$;
$EnQueue(Q, (i, m, n))$ for $\forall CL(m) = CL(n) = i, m < n$;
while $QueueEmpty(Q) == False$
  $DeQueue(Q, (i, m, n))$;
  if $CL(m) == CL(n)$
    $j = argmin_{l \in \{m,n\}} d(X_i, Y_l)$;
    $CL(j) = argmax_{l \in \{1, \cdots, k\} - \{i\}} d(X_l, Y_j)$;
    $EnQueue(Q, (i', m', j))$ for $\forall CL(m') = CL(j) = i'$
    $m' < j$;
    $EnQueue(Q, (i', j, n'))$ for $\forall CL(j) = CL(n') = i'$
    $j < n'$;
Update each Y_j's cluster label as $CL(j)$.

vices through links. For example, some "IT" company currently provides an early service of "Datacenter" but doesn't support a late service of "Cloud Computing". If there exists a direct connection between "Datacenter" and "Cloud Computing", then it is very possible that "Cloud Computing" will be provided by this company in the future. On the other hand, a provider may also transfer its influence to services provided by it. For instance, "IBM" provides both "Datacenter" and "Business Strategy" so that these two services have an indirect connection through the common provider "IBM". The spread of service influence resembles the heat diffusion phenomenon. Early choice of a provider (or a service) with many peers and services (or providers) in a cluster may act as heat sources, transfer its heat to its peers and services (or providers) and diffuse their influences to other majority. Finally, at a certain time, heat is diffused to the margin of the service network.

We use our unified random walk distance as the heat diffusion kernel since it captures the service (or provider) influence through both direct and indirect links. The heat diffusion

kernel H is a submatrix of R in Eq.(6.11).

$$H = \begin{bmatrix} R_{SS} & R_{SP} \\ R_{PS} & R_{PP} \end{bmatrix} \tag{6.14}$$

We utilize the matched clusters of services and providers from the disjoint clustering process as the heat (influence) source. A $(|S|+|P|)\times k$ initial heat matrix $sc(0)$ represents the heat source from which the heat kernel starts its diffusion process. The initial heat column vector $sc(0)(:, j)(j = 1, \cdots, k)$ of the service network at time 0 is defined below.

$$sc(0)(:, j) = (p_{1j}, \cdots, p_{|S|j}, p_{(|S|+1)j}, \cdots, p_{(|S|+|P|)j})^T \tag{6.15}$$

where p_{ij} is the probability of the i^{th} $(1 \leq i \leq |S|)$ service vertex in cluster S_j or the $(i-|S|)^{th}$ $(|S|+1 \leq i \leq |S|+|P|)$ provider vertex in cluster P_j. Due to the disjoint clustering, each initial p_{ij} is equal to 0 or 1 and $\sum_{j=1}^{k} p_{ij} = 1, \forall i$. Thus, $sc(0)$ is given as follow.

$$sc(0) = [sc(0)(:, 1), sc(0)(:, 2), \cdots, sc(0)(:, k)] \tag{6.16}$$

where $sc(0)$ essentially denotes the clustering distribution of services and providers after the clustering matching process.

We treat each $p_{ij} > 0$ as an initial influence source to execute the service influence propagation in each cluster until convergence, i.e., influence equilibrium. We argue that a service's partitioning not only depends on the cluster label of peer services but also lies on the cluster label of its providers. During the following influence propagation, a service continuously obtains the influences from its providers and peer services about the cluster

selection until it arrives at the margin of the service network.

$$\begin{aligned} sc(1) &= H \times sc(0) \\ &\dots \\ sc(t) &= H \times sc(t-1) \end{aligned} \tag{6.17}$$

Therefore, the service network's thermal capacity at time t, denoted by $sc(t)$, is defined as follow.

$$sc(t) = H^t \times sc(0) \tag{6.18}$$

where $sc(t)$ corresponds to a probabilistic clustering result, i.e., each entry $sc(t)(i, j)$ represents the probability of the i^{th} $(1 \leq i \leq |S|)$ service or the $(i - |S|)^{th}$ $(|S| + 1 \leq i \leq |S| + |P|)$ provider in the j^{th} cluster. We then normalize each $sc(t)(i, j)$ so that $\Sigma_{l=1}^{k} sc(t)(i, l) = 1$.

$$sc(t)(i, j) = \frac{sc(t)(i, j)}{\Sigma_{l=1}^{k} sc(t)(i, l)} \tag{6.19}$$

6.6 Service Influence Based Ranking

When ranking services over the global service network without considering which clusters they belong to, it is not clear to the user, from where to start looking at the results, since users usually prefer seeing a ranked result list in some area what they are interested in rather than a global ranked list. We require ranking functionality to present services to the user in a meaningful way i.e. by ranking them in each cluster based on a good clustering result. However, it is still not enough for a local ranking method to help the user efficiently distinguish the returned service list. We argue that the influence propagation among services and providers can provide more informative views of ranking result. The ranking of an

early service usually influences other late peer services through the direct connection. On the other hand, a provider may transfer its influence to the ranking of services provided by it. Their influences are finally diffused to the entire service network through the influence propagation.

The following heuristics rules give us initial ideas.

- Highly ranked services in each cluster have connections with many highly ranked services within the same cluster.
- Highly ranked services in each cluster are provided by many highly ranked providers within the same cluster.
- Highly ranked providers in each cluster own many highly ranked peer providers within the same cluster.
- Highly ranked providers in each cluster provide many highly ranked services within the same cluster.

We still utilize H in Eq.(6.14) as the propagating heat-diffusion kernel in the ranking process since it captures the direct connections between different types of objects based on the above four ranking rules. We use a $(|S|+|P|)\times k$ influence matrix $sr(0)$ to represent the initial ranking of services and providers in each cluster. The initial ranking score of the i^{th} object vertex in j^{th} cluster is defined below.

$$sr(0)(i,j) = \begin{cases} \Sigma_{l=1,l\neq i,sc(t)(l,j)>0}^{|S|+|P|} H(i,l), \ if \ sc(t)(i,j) > 0 \\ 0, \qquad otherwise \end{cases} \tag{6.20}$$

where $sr(0)(i,j)$ is the sum of the unified random walk distance from the i^{th} $(1 \leq i \leq |S|)$ service or the $(i-|S|)^{th}$ $(|S|+1 \leq i \leq |S|+|P|)$ provider to other objects partitioned into cluster $sc(t)(:,j)$. If a service (or a provider) in a cluster is connected to many peers and many providers (or services) in the same cluster, then it will achieve a higher initial ranking

score in this cluster. We then normalize each $sr(0)(i, j)$ so that $\Sigma_{l=1}^{|S|+|P|} sr(0)(l, j) = 1$.

$$sr(0)(i, j) = \frac{sr(0)(i, j)}{\Sigma_{l=1}^{|S|+|P|} sr(0)(l, j)} \tag{6.21}$$

Thus, we use the influence propagation to iteratively refine ranking scores based on the above heuristics rules.

$$\begin{gathered} sr(t)(:, j) = H \times \begin{bmatrix} sr(t-1)(1 : |S|, j) \\ sr(t-1)(|S|+1 : |S|+|P|, j) \end{bmatrix}, \\ sr(t)(i, j) = \frac{sr(t)(i, j)}{\Sigma_{l=1}^{|S|+|P|} sr(t)(l, j)} \end{gathered} \tag{6.22}$$

where $sr(t)(:, j)$ specifies the ranking score of each service (or provider) in cluster $sc(t)(:, j)$ at time t. During the influence propagation, a service's ranking is continuously refined by the ranking of its direct peers and providers as well as the ranking of these neighbors' neighbors. When the influence propagation reaches equilibrium, the final ranking score of a service in a cluster is jointly decided by the ranking scores of the services and providers in the same cluster which have at least one feasible path to this service.

The clustering and ranking based similarity is defined as follow.

$$\begin{gathered} e_{ij} = 1 - \frac{\sum_{l=1}^{k} |sc(t)(i, l)sr(t)(i, l) - sc(t)(j, l)sr(t)(j, l)|}{\sum_{l=1}^{k} sc(t)(i, l)sr(t)(i, l) + sc(t)(j, l)sr(t)(j, l)}, \\ \forall i, j \in \{1, \ldots, |S| + |P|\} \end{gathered} \tag{6.23}$$

We then substitute this new e_{ij} for the old one in T and recalculate the matrices T and R to further improve the performance of both clustering and ranking.

By assembling different parts, our ranking algorithm, ServiceRank, is presented in Algorithm 9. Steps 1 and 2 figure out a local optimal weight assignment for various types of links. Steps 3 calculates the transition probability matrix T and the unified random walk distance matrix R. Step 4 performs the clustering method, k-Medoids, to produce disjoint clusters of services and providers. The clustering matching process is then executed be-

Algorithm 9 ServiceRank

Input: a heterogeneous service network G, a length limit l of random walk paths, a restart probability c, a cluster number k.

Output: service ranking $sr(t)(:, j)$ in each cluster.

1: $\alpha_{SS} = \alpha_{SP} = \alpha_{PS} = \alpha_{PP} = \beta_1 = \ldots = \beta_m = 1.0$;
2: Generate the local optimal weights through the algorithm in [137];
3: Calculate T in Eq.(8.1) and R in Eq.(6.11);
4: Execute k-Medoids to produce clusters $S_1, \ldots, S_k$,
5: $P_1, \ldots, P_k$;
6: Match $S_1, \ldots, S_k$ to $P_1, \ldots, P_k$ through Algorithm 8;
7: Repeat until convergence:
8: Get probabilistic clusters $sc(t)(i, j)$ in Eq.(6.19);
9: Get local ranking $sr(t)(i, j)$ in Eq.(6.22);
10: Update all e_{ij}s in Eq.(6.23);
11: Recalculate T and R.

tween service clusters and provider clusters in step 5. Steps 7-10 repeatedly run the service influence based clustering and ranking and iteratively update the clustering and ranking based similarity so that T and R are continuously refined to generate better ranking result until convergence.

6.7 Experimental Evaluation

We have performed extensive experiments to evaluate the performance of ServiceRank on both synthetic and real service datasets.

6.7.1 Experimental Datasets

BSBM Dataset: we modify the BSBM data generator [139] and create a dataset with 246, 161 triples where "Provides" is used to model the relationship between "Service" and "Provider"s providing them, while an instance of "Service" has multiple instances of properties "Capability", "Function" and "Type", and an instance of "Provider" contains multiple instances of properties "Feature" and "Type". There are totally 10, 000 "Service" instances and 3, 628 "Provider" instances with 10 "Type" instances and 5 instances of "Capability", "Function" and "Feature", respectively.

IBM Service Dataset: the dataset contains a total of 41, 292 triples with 2, 450 "Ser-

vice" instances and 928 "Provider" instances. Each "Service" instance may have three types of properties: "Type", "Category" and "Capability", respectively. On the other hand, each "Provider" instance may contain two kinds of properties: "Type" and "Category". We build a service network where object vertices represent services or providers, attribute vertices denote their properties, object edges represent the relationship between object vertices, attribute edges specify the relationship between object vertices and their associated attributes.

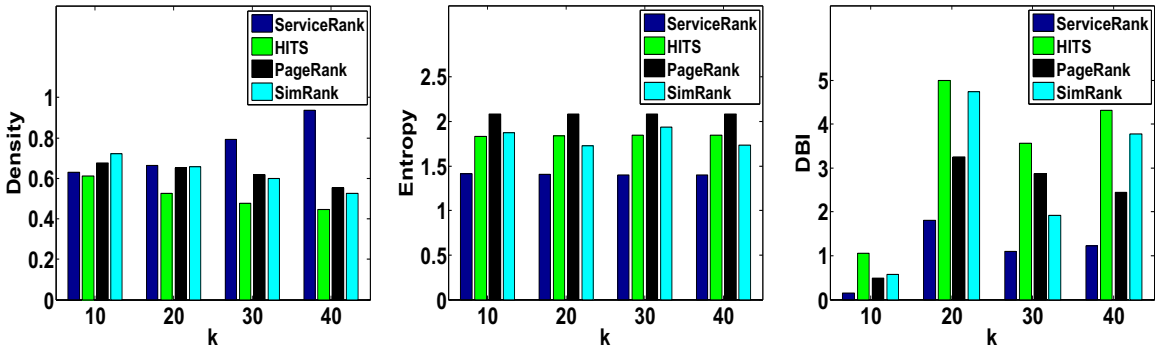
6.7.2 Comparison Methods and Evaluation

We compare **ServiceRank** with three representative ranking algorithms, **HITS** [130], **PageRank** [96] and **SimRank** [134]. Since the last three ranking algorithms can not directly handle a heterogeneous network, we treat services, providers and attributes as homogeneous vertices with the same type and rank homogeneous vertices in each cluster. We choose three disjoint clustering methods of k-Means [138], k-Means++ [140] and k-Medoids [25] for these three ranking algorithms, respectively. ServiceRank integrates various types of entity, link and attribute information into a unified distance measure with the local optimal weights on a heterogeneous service network. Service influence based clustering and ranking are iteratively performed to mutually enhance the performance of both of them.

Evaluation Measures We use three measures of density, entropy and Davies-Bouldin Index (DBI) to evaluate the quality of clusters $\{S_i\}_{i=1}^{k}$ and $\{P_i\}_{i=1}^{k}$ generated by different methods. The metrics are defined as follows.

$$density(\{S_i \cup P_i\}_{i=1}^{k}) = \sum_{i=1}^{k} \frac{\sum_{v_p, v_q \in S_i \cup P_i, (v_p, v_q) \in E} min(\pi_{pi}, \pi_{qi})}{|E|} \tag{6.24}$$

where π_{pi}, π_{qi} represent the probabilities of vertex v_p and vertex v_q in the i^{th} cluster respectively. $min(\pi_{pi}, \pi_{qi})$ is equal to 1 for disjoint clustering methods but may be less than 1 for



(a) density (b) entropy (c) DBI

Figure 6.2: Cluster Quality Comparison on BSBM 13,628 Vertices

our probabilistic clustering.

$$entropy(\{S_i \cup P_i\}_{i=1}^{k}) = \sum_{i=1}^{m} \frac{\beta_i}{\sum_{l=1}^{m} \beta_l} \sum_{j=1}^{k} \frac{|S_j|}{|S|} entropy(a_i, S_j) + \frac{|P_j|}{|P|} entropy(a_i, P_j) \tag{6.25}$$

where $entropy(a_i, S_j) = -\sum_{l=1}^{|S|} p_{ijl} log_2 p_{ijl}$ and p_{ijl} is the percentage of services in cluster S_j which have value a_{il} on attribute a_i. $entropy(\{S_i \cup P_i\}_{i=1}^{k})$ measures the weighted entropy from all attributes over k clusters.

$$DBI(\{S_i \cup P_i\}_{i=1}^{k} = \frac{1}{2k} \sum_{i=1}^{k} max_{j \neq i}(\frac{d(c_S^j, c_S^i)}{\sigma_S^j + \sigma_S^i}) + max_{j \neq i}(\frac{d(c_P^j, c_P^i)}{\sigma_P^j + \sigma_P^i}) \tag{6.26}$$

where c_S^x $(1 \leq x \leq k)$ is the centroid of cluster S_x, $d(c_S^i, c_S^j)$ is the random walk distance between centroids c_S^i and c_S^j, and σ_S^x $(1 \leq x \leq k)$ is the average similarity of all elements in cluster S_x to centroid c_S^x. $entropy(a_i, P_j)$, c_P^x, $d(c_P^i, c_P^j)$ and σ_P^x corresponding to providers P have the similar meanings. A cluster with high intra-cluster similarity and low inter-cluster similarity will have a low DBI value.

6.7.3 Cluster Quality Evaluation

Figure 6.2 (a) shows the density comparison on BSBM 13,628 Vertices by varying the number of clusters $k = 10, 20, 30, 40$. The density values by ServiceRank obviously higher than that other three methods except $k = 10$. They remain in the range of 0.63 or above

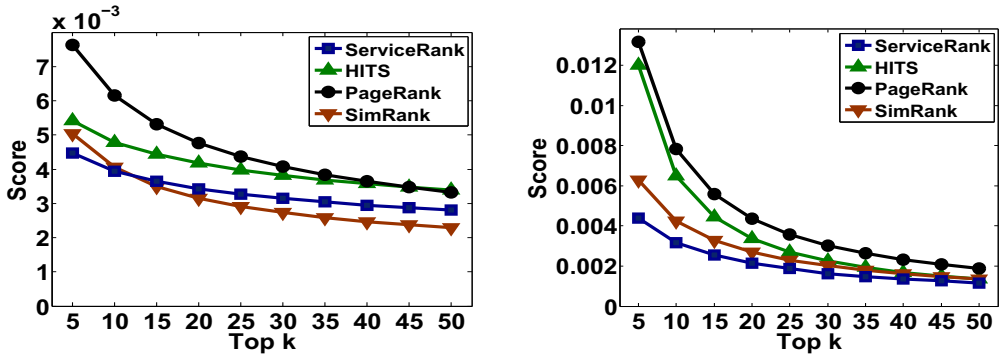
even when k is increasing. This demonstrates that ServiceRank can find densely connected components. Different from the disjoint clustering methods, the density value by our service influence based probabilistic clustering keeps increasing when k is increasing since each vertex is partitioned into more clusters with a larger k.

Figure 6.2 (b) presents the entropy comparison between four methods on BSBM $13,628$ Vertices. ServiceRank achieves the lowest entropy than other three methods. Entropy by ServiceRank is as low as less than 1.32 while entropy by other three approaches is still above 1.83 since they partitions the service network without considering different degrees of importance for multiple types of links. As shown in Figures 6.2 (a) and (b), the performance of ServiceRank is the best in terms of both density and entropy. This is because its distance function integrates the local optimal weight assignment for multiple types of structural and attribute links, thus it achieves a good performance on both criteria. On the other hand, other three methods do not differentiate the links between different kinds of objects. It is equivalent to combine multiple types of structural and attribute similarities with the equal weighting factor of 1, which usually does not produce a good clustering.

Figures 6.2 (c) shows the DBI comparison on BSBM $13,628$ Vertices with different k values by four methods. ServiceRank has the lowest DBI of around $0.15 - 1.80$, while PageRank, HITS and SimRank have a much higher DBI than ServiceRank. This demonstrates that ServiceRank can achieve both high intra-cluster similarity and low inter-cluster similarity.

6.7.4 Ranking Quality Evaluation

Figures 6.3 (a) and (b) plot the average ranking scores of top-k vertices in each cluster on two datasets, respectively. The score curve of ServiceRank with respect to k value is most stable among four ranking algorithms. Especially, the curve is almost a stable horizontal line when the k value stands in between 20 and 50. This is because ServiceRank achieves both the highest intra-cluster similarity and the lowest inter-cluster similarity. In addition,

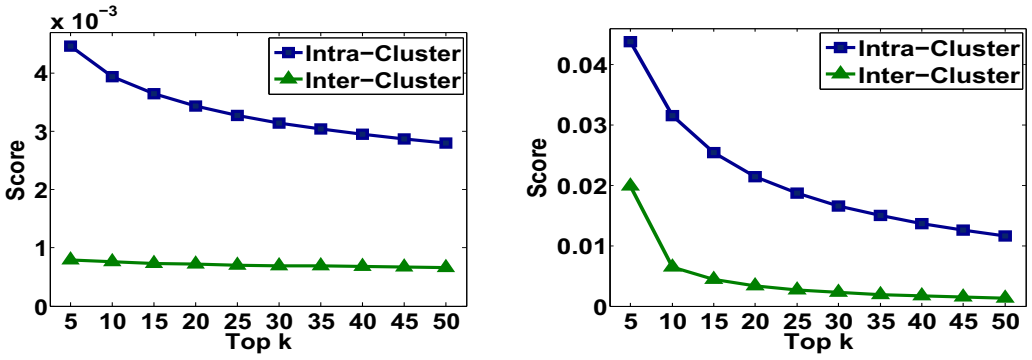


(a) density (b) entropy

Figure 6.3: Ranking Quality Comparison

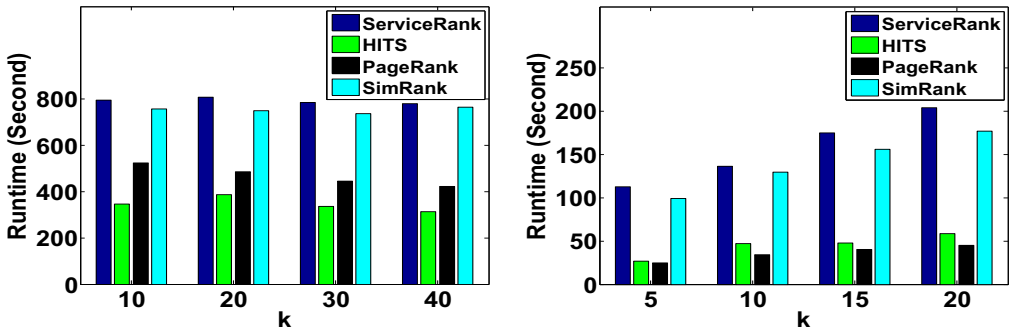
the iteratively updated ranking-based similarity between services makes the generated clusters get more cohesive intra-cluster structure and more homogeneous vertex properties. As a result, services within clusters are connected to similar services, providers and attributes so that they achieve similar ranking scores in terms of our heuristics ranking rules. The quality by PageRank and HITS is the worst since the resulted clusters without considering the weight assignment and the alternative iterations of clustering and ranking have a rather random distribution of services in terms of both structure and attribute similarities. Although SimRank also doesn't consider the weight assignment and the ranking iterations, the quality of SimRank stands in between since it iteratively updates the pairwise similarities.

Figures 6.4 (a) and (b) present the average ranking scores of services within clusters and services outside clusters by ServiceRank, respectively. We rank all services in each cluster and treat a service with a positive probability in cluster as the service within cluster. The score curve of services within clusters is much higher than that of services outside clusters since services within clusters are connected to many services, providers and attributes in clusters but there are few links between cluster and services outside cluster. During each iteration of clustering and ranking, the clustering iteratively refines the weights of multiple types of links in terms of their contribution. The updated ranking-based similarity with the refined weights makes services with the similar cluster distribution and the similar ranking scores become more similar so that they will have a higher probability to be partitioned



(a) density (b) entropy

Figure 6.4: Ranking Accuracy Comparison



(a) BSBM 13,628 Vertices (b) IBM 3,652 Vertices

Figure 6.5: Efficiency Evaluation

into the same clusters in the next iteration.

6.7.5 Efficiency Evaluation

The running time by each algorithm for BSBM $13,628$ Vertices and IBM $3,378$ Vertices is summarized in Figures 6.5 (a) and (b), respectively. As we can observe, HITS and PageRank are the most efficient as they execute both clustering and ranking algorithms only once on the homogeneous service network (i.e., without running the clustering matching and the alternative iterations of clustering and ranking). SimRank is usually $1.5 - 4$ times slower than HITS and PageRank since it needs to iteratively update the pairwise similarities on the entire service network. Although ServiceRank does not need to update the pairwise similarities, it is about $1.05 - 1.16$ times slower than SimRank. As ServiceRank needs to iteratively adjust the weights of multiple types of structure and attribute links, it itera-

tively computes the random walk distance matrix from scratch on the service network. In addition, service influence based clustering and ranking are iteratively performed so that the total cost of ServiceRank has been greatly increased. Although ServiceRank is more expensive, the iterative refinement improves the ranking quality a lot, as demonstrated in Figures 6.3 and 6.4.

6.8 Conclusion

In this chapter, we have presented a unified neighborhood random walk distance measure integrating various types of link and attribute information with the local optimal weight assignment on a heterogeneous service network. We present a reinforcement algorithm that is developed to tightly integrate ranking and clustering by mutually and simultaneously enhancing each other such that the performance of both can be improved. We propose an additional greedy strategy to efficiently execute clustering matching process to align clusters for each type of objects in the heterogeneous service network. Our extensive evaluation on both synthetic and real service networks demonstrates the power of our method in terms of the quality of both clustering and ranking.

CHAPTER 7

GRAPHLENS: MINING ENTERPRISE STORAGE WORKLOADS USING GRAPH ANALYTICS

Conventional methods used to analyze storage workloads have been centered on relational database technology combined with attributes-based classification algorithms. This chapter presents a novel analytic architecture, GraphLens, for mining and analyzing real world storage traces. The design of our GraphLens system embodies three unique features. First, we model storage traces as heterogeneous trace graphs in order to capture diverse spatial correlations and storage access patterns using a unified analytic framework. Second, we employ and develop an innovative graph clustering method to discover interesting spatial access patterns. This enables us to better characterize important hotspots of storage access and understand hotspot movement patterns. Third, we design a unified weighted similarity measure through an iterative learning and dynamic weight refinement algorithm. With an optimal weight assignment scheme, we can efficiently combine the correlation information for each type of storage access patterns, such as random v.s. sequential, read v.s. write, to identify interesting spatial correlations hidden in the traces. Extensive evaluation on real storage traces shows GraphLens can provide scalable and reliable data analytics for better storage strategy planning and efficient data placement guidance.

7.1 Introduction

Performance optimization in enterprise storage systems has primarily relied on the ability to isolate and control workloads that were relatively well understood [141, 142, 143, 144]. With virtualized environments and cloud implementations, enterprise storage systems have to support a mixture of a large number of disparate workloads from a variety of applications [145, 146, 147]. Thus, storage systems not only need to deal with changes within a

single workload, but also have to respond to changes to the workload mix, enabling more efficient sharing of the underlying infrastructure. Although Flash, DRAM and newer high performance storage hardware hold the promise to alleviate the performance problem, to fully capitalize on the potential of these new storage devices, intelligence and automation are required to identify the right data to be placed on the right storage devices or storage tiers at the right time.

Trace analysis is recognized as a viable model to assist with characterizing workloads and gaining deeper insights into workload behavior. Conventional storage trace analysis is primarily carried out by the per-column based statistical analysis (single attribute based access pattern) or the row-based statistical analysis using vector similarity. Characterizing workloads in depth and from different levels of granularity of spatial dimension is challenging. The challenge can be more demanding when a single volume represents a varying mix of workloads and such workload mix may change over time. We argue that understanding similarity and causality of access patterns can offer many opportunities for optimization of performance such as intelligent data placement.

This chapter presents a storage trace analysis architecture, GraphLens, for mining and analyzing real world storage traces. By innovative exploration of graph analytics, GraphLens offers three original contributions. First, storage traces are modeled as heterogeneous trace graphs in order to use a unified analytic model to study the complex spatial correlations among storage addresses at different levels of granularity in terms of their access patterns. Second, an innovative graph clustering method is developed to discover interesting spatial correlations by clustering storage addresses with a dynamic weighting scheme that continuously refines the weights on different access patterns of the storage addresses towards clustering convergence. This allows us to identify deeper spatial correlations among storage addresses beyond direct neighboring addresses. Third, our weight assignment scheme can efficiently combine the correlation information for each type of storage access patterns, such as random v.s. sequential, read v.s, write, to identify hotspots and their

movements along diverse spatial dimensions of the trace. Extensive evaluation on three real storage traces demonstrates that GraphLens can perform deep trace analysis to derive new insights and new values for better storage planning and more efficient data placement strategies.

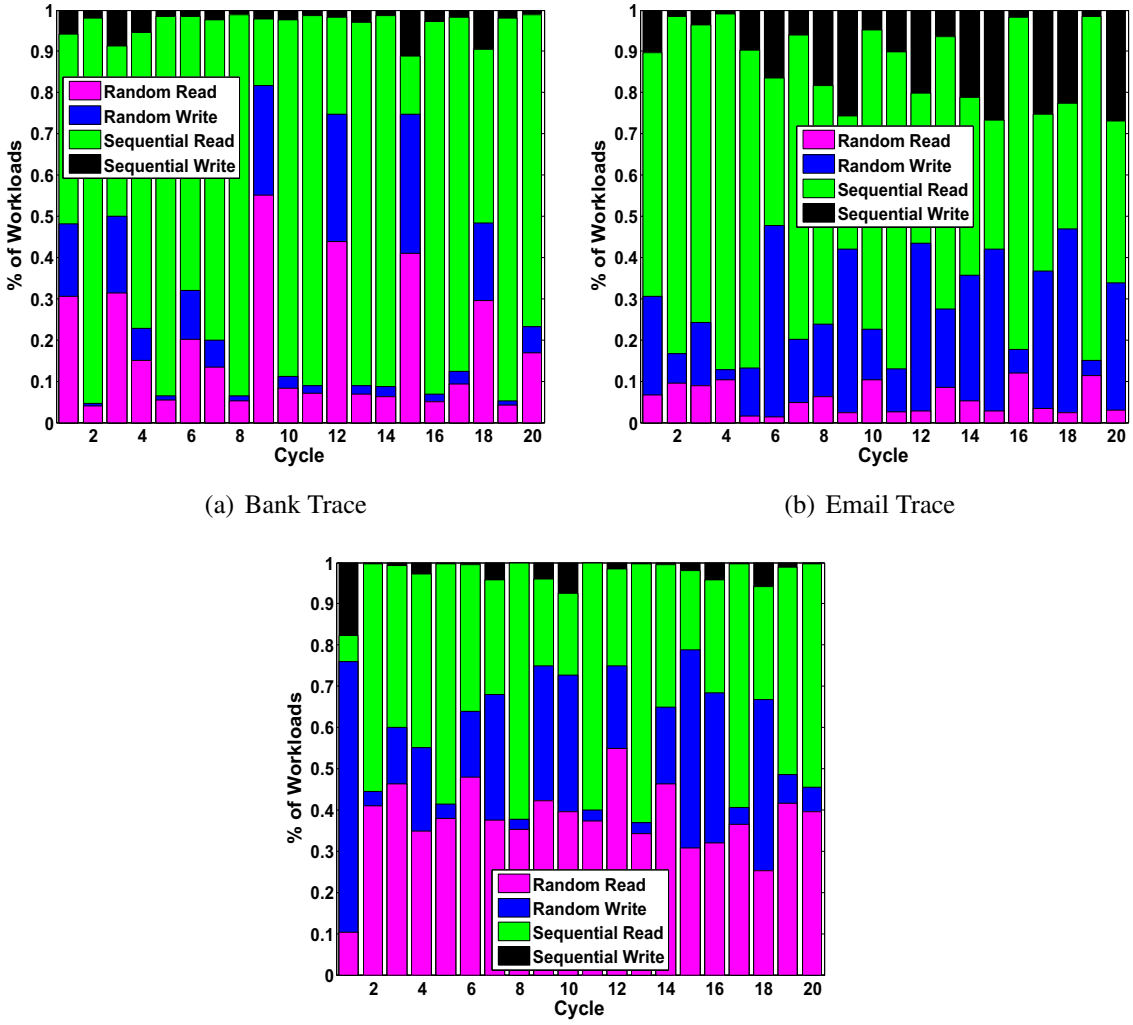
7.2 Motivation and Background

Related Work. Storage traces of production servers are valuable and critical in gaining insights on design, implementation and optimization of both modern storage servers and I/O intensive applications. However, mining and analyzing storage access patterns from real world workload traces has been scarce and superficial for a number of reasons, including difficulty in obtaining traces of production servers in diverse domains and absence of effective trace analysis models and algorithms that can infer deeper insight from limited traces of production systems. More seriously, many past trace-based studies have predated technology trends [148]. In the last decade, there are only a few studies [149, 141, 148, 150, 151] have dedicated to developing methodologies for characterization of real world workload traces. [149] analyzed four storage workload traces of production Windows servers with respect to block level requests, file access frequencies and read/write ratios. It performs trace analysis by measuring the spatial and temporal self-similarity based on variance and mean of storage log data. The approach in [144] assumes that workloads are well defined and can be cleanly isolated in order to train a classifier to identify workload phases using supervised learning. In addition, [148] studied the same large scale network file system workloads as reported in [150] using a multi-dimensional trace analysis methodology instead of single dimension based method. However, none of existing work has analyzed workload traces of production storage servers based on graph analytics. A unique advantage of modeling storage traces using graphs is the ability to conduct deep-analytics on both self-similarity and neighborhood similarity from both spatial and temporal perspectives. More importantly, GraphLens derives insights from traces with no

assumption of a priori knowledge about workloads. Graph as an expressive data structure is popularly used to model structural relationship between objects in many application domains, ranging from web, social networks, biological networks to sensor networks [105, 96, 130, 11, 99, 51, 152, 13, 20, 28]. However, to the best of our knowledge, GraphLens is the first one that applies and extends graph mining to storage trace analysis. A unique feature of GraphLens is its ability to effectively identify fine-grained behavioral similarity across spatial and temporal dimensions using storage block traces.

Workload Traces. We analyze block-level traces collected from three large enterprise storage installations: a live banking environment, a retail backend system environment and an email server environment. Each of the traces consists of storage workloads collected over every 15 minute period (referred to as "cycle") for storage addresses, called "extents", each extent representing 1 GB logical address unit. The traces provide summary information on the number of random read, random write, sequential read and sequential write IO accesses over one week period (7 days). The only knowledge we know about each of these environments is that multiple workloads (such as applications and backup) may have been executing simultaneously. But we have no details regarding the exact nature of the workloads.

Figures 7.1 (a), (b) and (c) exhibit the distribution four access patterns observed on the three real world storage traces. For ease of presentation, we group the total of 2010 cycles into 20 cycle groups and summarize the access account for each extent in each cycle group. Figure 7.1 (a) presents the access activities on Bank Trace and "sequential read" is obviously the dominating access pattern. However, the access activities on Email Trace (Figure 7.1(b)) are often dominated by "sequential read" access pattern, with "random write" and "sequential read" as secondary behavior. In contrast, the workloads on Store Trace is mainly dominated by "random read" access pattern. We argue that by analyzing spatial, temporal and hot spot correlations from block-level traces we can provide broader and deeper insights for better tradeoffs in storage system design and implementation.



(a) Bank Trace

(b) Email Trace

(c) Store Trace

Figure 7.1: IO Workloads by Four Access Patterns

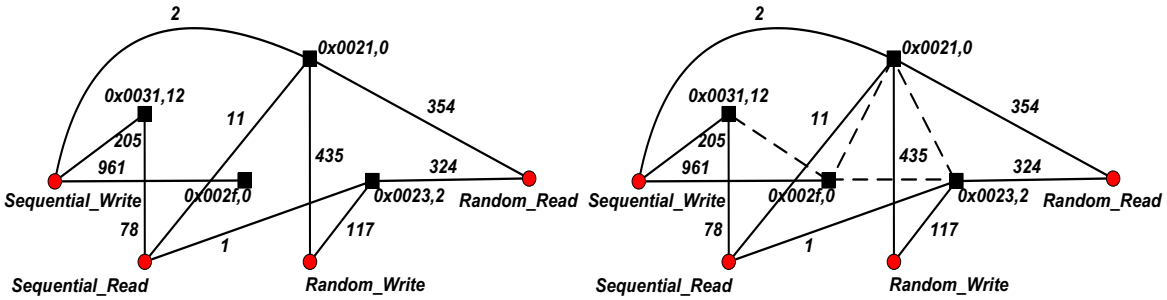
7.3 Overview

GraphLens by design aims at exploiting graph data analytic techniques on multi-dimensional storage traces to derive deep insights hidden in the storage logs, such as spatial access correlation and hot-spot dynamics. For example, how are different addresses accessed similarly within a cycle or amongst cycles? how does the access pattern of a storage address change between cycles? do spatial patterns interact with one another? what types of spatial access patterns are common in real world traces? and how hot spots move across extents (spatial)?

One approach for discovering and mining interesting correlations is to associate differ-

Table 7.1: A Sample Trace for a Cycle

Lun	Extent	RR	RW	SR	SW
0x0021	0	354	435	11	2
0x0023	2	324	117	1	0
0x002f	0	0	0	0	961
0x0031	12	0	0	78	205



(a) Graph Model of a Single Cycle

(b) Structure Relationship Deriving

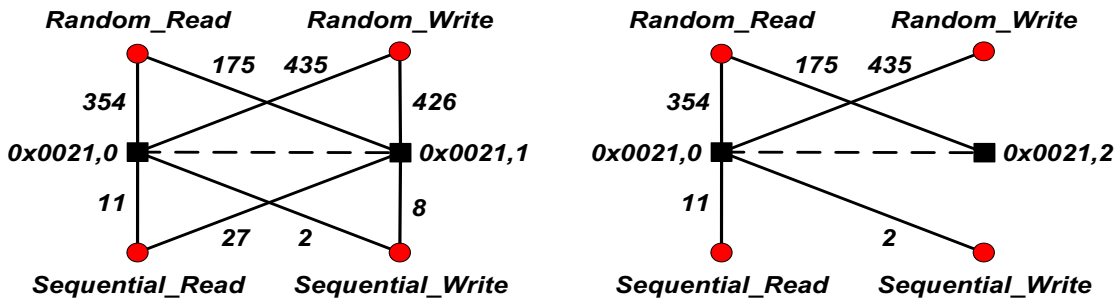
Figure 7.2: An Illustrating Example of Heterogeneous Trace Graph

ent storage addresses by utilizing their common attributes (access patterns), such as random v.s. sequential access and read v.s. write access. This motivates us to introduce two levels of abstractions for analyzing storage traces. First, we model storage traces as heterogeneous graphs. Second, we employ innovative graph analytic methods to measure and discover correlations among storage addresses. This two-level abstraction enables us to study the access correlations among different addresses by examining two types of vertices: structure vertices representing storage addresses (Lun (Volume), Extent) and attribute vertices (access patterns such as random, sequential, read or write) and their explicit and implicit relationships. Compared to naive vector-based correlation (record by record comparison), modeling storage access logs as a graph allows us to observe and understand not only those shallow correlations reflected from direct relationship between an address and its access pattern (attribute) but also enables us to derive deeper correlations that can only be inferred through reasoning over both direct and indirect correlations in a probabilistic manner.

7.3.1 Modeling Traces as Graphs

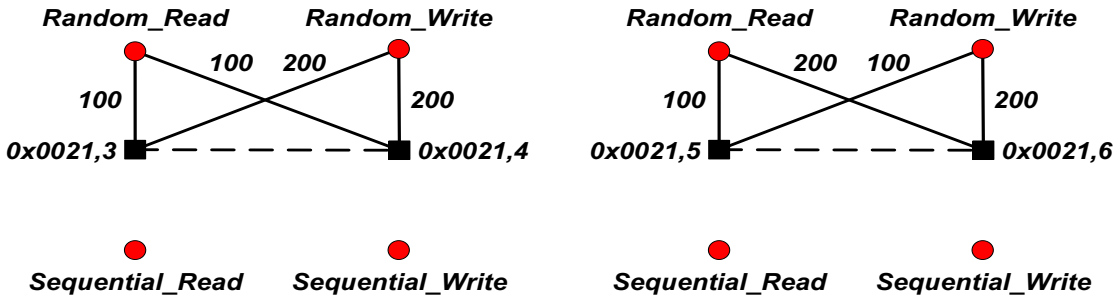
An enterprise storage trace recorded with multiple cycles is logged with a set of attributes (access patterns), such as random read (RR), random write (RW), random transfer (RT), sequential read (SR), sequential write (SW), sequential transfer (ST), etc. We model each cycle t_i of the storage log as **heterogeneous trace graph**, denoted as $G_i = (V, A, E_i, F_i)$, where $n = |V|$ specifies the size of the structure vertex set, $m = |A|$ defines the size of attribute vertex set in the trace, E_i denotes a set of structure edges between structure vertices and F_i represents the set of m types of attribute edges between V and A or between A and V. $v \in V$ is a structure vertex, representing a storage address and $a \in A$ denotes an attribute vertex associated to a structure vertex v, specifying an associated attribute of the address v. A structure edge $e \in E_i$ connects two structure vertices and an attribute edge $f \in F_i$ represents the relationship between a structure vertex and its associated attribute, weighted by the frequency of the corresponding access pattern within the given cycle. The initial heterogeneous graph G_i for each cycle t_i is a bipartite graph with only attribute edges. By employing GraphLens, we learn the correlations between structure vertices via their associated attributes. For instance, the more the access patterns shared by two addresses are, the greater the similarity between two addresses is.

Table 7.1 presents an example of a real storage trace. Each combination of Lun and Extent represents a unique storage address. RR, RW, SR, and SW correspond to four kinds of access patterns: random read, random write, sequential read, and sequential write, respectively. Figure 7.2 (a) is the heterogeneous graph representation of the sample trace in Table 7.1. This trace graph has heterogeneous vertices: structure vertices (black square) represent the storage addresses, attribute vertices (red circle) specify 4 types of attributes: RR, RW, SR and SR. In addition, this trace graph has explicit attribute edges (solid lines), each representing a relationship between an structure vertex and one of its four types of attributes, and derived relationships (dashed lines) that represent the spatial correlation between two structure vertices, as shown in Figure 7.2 (b). Although the four address



(a) More Paths (b) Less Paths

Figure 7.3: Summarization of all Possible Paths



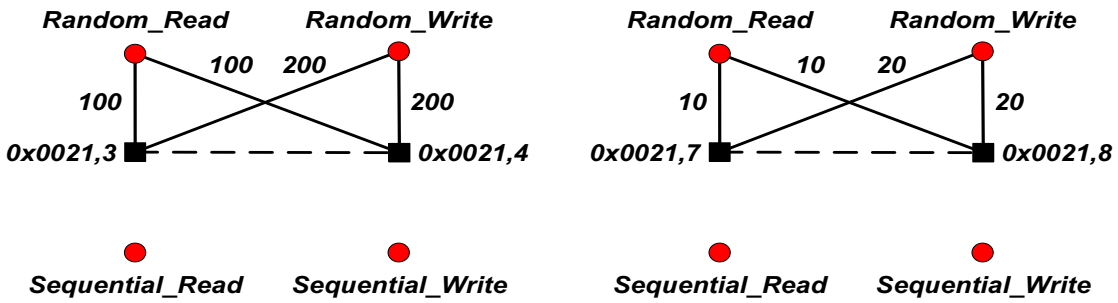
(a) Weight Match (b) Weight Mismatch

Figure 7.4: Attribute Weight Match

vertices have no direct correlations, we can learn the spatial correlations among different structure vertices because they can be reached by traversing the graph via attribute vertices.

Case 1: summarization of all paths between any pair of extents. In Figure 7.3 (a), there exists four 2-hop paths between extents "0x0021, 0" and "0x0021, 1" through four attribute vertices respectively. In comparison to Figure 7.3 (b), there is only one 2-hop path between two extents through RR. We make the following observation: extents "0x0021, 0" and "0x0021, 1" are more similar than extents "0x0021, 0" and "0x0021, 2" since there are more reachable paths between the first two extents.

Case 2: attribute differentiation by attribute weight match. For both Figure 7.4 (a) and Figure 7.4 (b), two extents are reachable by two 2-hop paths through RR and RW respectively. However, the two addresses in Figure 7.4 (b) on each of RR and RW have diverse edge weights. Thus, extent "0x0021, 3" and extent "0x0021, 4" are more similar



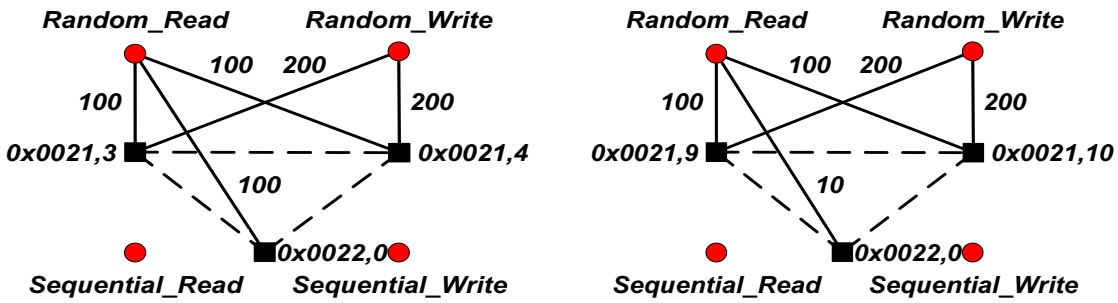
(a) Significant Weights (b) Insignificant Weights

Figure 7.5: Attribute Weight Significance

than extent "0x0021, 5" and extent "0x0021, 6" because the first pair of extents not only have the same access patterns (RR and RW) but also have the same access counts (100 for RR and 200 for RW).

Case 3: attribute differentiation by attribute weight significance. In both Figure 7.5 (a) and Figure 7.5 (b), two extents are reachable by two 2-hop paths through RR and RW respectively and the corresponding attribute edge weights are the same respectively. However, the corresponding attribute edge weights (100 for RR and 200 for RW) in Figure 7.5 (a) are larger than that (10 for RR and 20 for RW) in Figure 7.5 (b). Thus, two extents in Figure 7.5 (a) are more similar than two extents in Figure 7.5 (b).

Case 4: summarization of all possible k-hop paths between pairwise extents. In the above cases, we only consider 2-hop paths between extents, i.e., direct relationships between extents. However, we should consider all possible k-hop paths, i.e., both direct and indirect relationships, to achieve a comprehensive and fair comparison result when we calculate the similarity scores between two extents. The only difference between Figures 7.6 (a) and (b) is the attribute edge weights between extent "0x0022, 0" and access pattern RR. Note that there is a 4-hop path between extent "0x0021, 3" (or "0x0021, 9") and extent "0x0021, 4" (or "0x0021, 10"): "0x0021, 3" (or "0x0021, 9") → "Random_Read" → "0x0022, 0" → "Random_Read" → "0x0021, 4" (or "0x0021, 10"). We argue that extents "0x0021, 3" and "0x0021, 4" in Figure 7.6 (a) have larger attribute edge weights and thus are more similar than extents "0x0021, 9" and "0x0021, 10" in Figure 7.6 (b). The



(a) Significant 4-hop Path

(b) Insignificant 4-hop Path

Figure 7.6: k-hop Path

similarity between two extents depends on not only their direct relationships (their own access patterns and access counts) but also their indirect relationships (predecessors' and successors' access patterns and access counts).

7.3.2 Trace Analysis with GraphLens

GraphLens performs trace analysis to derive deep insights on spatial/temporal access correlations and hotspot characterization in two phases: (1) extent similarity computation and (2) spatial based graph clustering.

In Phase I, we measure pairwise extent similarity within a cycle in terms of two factors: how similar their access patterns (attribute) are (direct correlation) and how similar their k-hop neighbor extents are in terms of their access patterns. In order to capture the spatial access similarity between storage addresses in terms of both direct and indirect correlations, we introduce a **unified weighted extent neighborhood random walk similarity measure**. It is used to measure the closeness between extents based on all four types of attribute edges, each with an initial weight. This unified similarity measure captures the connectivity and the vicinity between extents (structure vertices).

In Phase II, we utilize this unified similarity measure to cluster all extents in trace graph G_i into k_i clusters with initial centroids and initial weights. We employ a dynamic weight tuning method combined with an iterative refinement mechanism of centroid update and vertex assignment to quantitatively estimate the importance of various types of attributes

and attribute links in terms of their contribution to the clustering process. Formally, given a heterogeneous trace graph G_i for cycle t_i, the problem of **spatial extent clustering** is to partition the objective extents V into k_i disjoint clusters $C_1, C_2, \ldots, C_{k_i}$, where $i \in \{1, 2, \ldots, N\}$, N is the number of cycles, $V = \bigcup_{p=1}^{k_i} C_p$ and $C_p \bigcap C_q = \phi$ for $\forall p, q, 1 \leq p, q \leq k_i, p \neq q$, to ensure: (1) the extents within each cluster have larger similarity scores, while the extents in different clusters have smaller similarity scores; and (2) the extents within clusters have low diversity on access patterns, while the extents in different clusters have highly diverse access patterns. The spatial based clustering can indicate where the hotspots are and how such hotspots move across extents (along spatial dimension).

7.4 Methodology

This section describes the two-phase correlation analysis in GraphLens: extent-similarity based spatial clustering.

7.4.1 A Unified Spatial Similarity Measure

In GraphLens, we propose to use a unified similarity measure based on the neighborhood random walk model to infer the spatial access correlations between extents and the temporal access correlation between cycles. In the heterogeneous graph, some vertices are close to each other while some other vertices are far apart based on connectivity. Random walk distance can accurately capture such pairwise vertex closeness. Recall the example in Figure 7.2, there exists a random walk path between two extents $v_1, v_2 \in V$ if (1) v_1 and v_2 have the same neighbor extent $v_3 \in V$; or if (2) v_1 and v_2 have the same attribute $a \in A$. If there are multiple random walk paths connecting v_1 and v_2, then they should be very close in terms of similar access patterns. On the other hand, if there are very few or no paths between v_1 and v_2, then they should be far apart in terms of diverse access patterns.

Definition 23 (Transition Probability) *Let V be the set of n extents, A be the set of m associated attributes, the transition probability matrix $P(i)$ of a heterogeneous graph G_i*

for cycle t_i is defined as follow.

$$P(i) = \begin{bmatrix} P_{SS} & P_{SA} \\ P_{AS} & P_{AA} \end{bmatrix} \tag{7.1}$$

where P_{SS} is an $n \times n$ matrix representing the transition probabilities between structure vertices; an $n \times m$ matrix P_{SA} specifies the transition probabilities from structure vertices to attribute vertices; P_{AS} denotes the transition probabilities from attribute vertices to structure vertices; and P_{AA} is an $m \times m$ matrix representing the transition probabilities between attribute vertices.

In the context of heterogeneous trace graph, submatrices P_{SS} and P_{AA} have all zero entries since there is no connection between structure vertices or between attribute vertices. However, the corresponding submatrices in the power of $P(i)$, such as $P(i)^2, P(i)^3, \ldots$, may contain non-zero elements since there may exist possible paths through other vertices.

To capture the fact that each type of attribute edges may have different degrees of contribution in random walk similarity, we assign an individual weight for each type of attribute edges. Initially, all weights are set to equal value, say 1.0. We design a dynamic weight tuning method to produce an optimal weight assignment for all types of links in the next section. Based on this weight assignment, each submatrix in $P(i)$ is defined as follow.

$$P_{SA}(p,q) = \begin{cases} \dfrac{\alpha_q e_{pq}}{\sum_{r=1}^{m_i} \alpha_r e_{pr}}, & if(v_p, a_q) \in F_i \\ 0, & otherwise \end{cases} \tag{7.2}$$

where e_{pq} represents the count of access pattern a_q that storage extent v_p has in the given cycle. For instance, a storage extent of "0x0021, 0" has the count of 354 on "random read" in the example cycle of Figure 7.2(a). α_q denotes the weight of attribute edges from any of the structure vertices to attribute vertex a_q. Since each row of transition probability matrix

should sum to 1, we employ the row-wise normalization for P_{SA}.

$$P_{AS}(p,q) = \begin{cases} \dfrac{\alpha_p e_{pq}}{\sum_{r=1}^{n_i} \alpha_p e_{pr}} = \dfrac{e_{pq}}{\sum_{r=1}^{n_i} e_{pr}}, & if(a_p, v_q) \in F_i \\ 0, & otherwise \end{cases} \tag{7.3}$$

where e_{pq} specifies the count of an access pattern a_p achieved by a storage extent v_q. α_p denotes the weight of attribute edges from attribute vertex a_p to structure vertices. Different from the normalization in P_{SA}, the count on pattern a_p by extent v_q is normalized by the counts on pattern a_p by all extents.

A random walk on a heterogeneous trace graph G_i is performed in the following way. Suppose a particle starts at a certain vertex v_0 and walks to a vertex v_s in the s^{th} step and it is about to move to one of the neighbors of v_s, denoted as $v_t \in N(v_s)$, with the transition probability $P(i)(s,t)$, where $N(v_s)$ contains all neighbors of vertex v_s. The vertex sequence of the random walk is a Markov chain. The probability of going from v_i to v_j through a random walk of length l can be obtained by multiplying the transition probability matrix l times.

Definition 24 (Unified Random Walk Similarity) *Let $P(i)$ be the transition probability of a heterogeneous trace graph G_i, l be the length that a random walk can go, and $c \in (0,1)$ be the restart probability, the unified random walk similarity $s(u,v)$ from vertex $u \in V \bigcup A$ to vertex $v \in V \bigcup A$ in G_i is defined as follow.*

$$s_i(u,v) = \sum_{\substack{\tau: u \rightsquigarrow v \\ length(\tau) \leq l}} p(\tau) c (1-c)^{length(\tau)} \tag{7.4}$$

where τ is a path from u to v whose length is $length(\tau)$ with transition probability $p(\tau)$ which is equal to the multiplication of the transition probability of each step in path τ. $s_i(u,v)$ reflects the extent closeness within cycle t_i based on multiple types of attribute information.

The matrix form of the unified random walk similarity is given as follow.

$$R(i) = \sum_{\gamma=1}^{l} c(1-c)^{\gamma} P(i)^{\gamma} \tag{7.5}$$

where an $(n + m) \times (n + m)$ matrix $R(i)$ sums over the dependency of all possible paths between two extents. Each entry $s_i(u, v)$ in $R(i)$ measures the similarity between extent vertex u and extent vertex v within cycle t_i.

7.4.2 Spatial Extent Clustering

Our extent clustering framework, E-Cluster, partitions extents in a heterogeneous trace graph G_i into k_i densely connected clusters. Due to space limit, we will briefly illustrate the extent clustering framework by focusing on different points. E-Cluster follows the traditional K-Medoids clustering method [25] by using the unified random walk similarity $R(i)$ with k_i extents of high degree as the initial centroids and the initial weights $\alpha_{11}^0, \ldots, \alpha_{im}^0$ as an input. At each iteration, based on unified extent random walk similarity scores, we select the most centrally located extent in each of the k_i clusters to obtain k_i new centroids, and assign the rest of extents to their closest centroids. The objective of clustering is to maximize intra-cluster similarity and minimize inter-cluster similarity. The weight update method computes the weighted contributions of each kind of attribute links to both clustering convergence and clustering objective, and updates m weights accordingly after each iteration. This process is repeated until convergence.

Thus the graph clustering problem can be reduced to three subproblems: (1) cluster assignment, (2) centroid update and (3) weight adjustment, each with the goal of maximizing the objective function. The first two problems are common to all partitioning clustering algorithms. Hence we only focus on the third subproblem, weight adjustment in this chapter. We employ a dynamic weight adjustment method to iteratively improve the spatial extent clustering objective. Let $\alpha_{ip}^t (p = 1, \ldots, m)$ be the weights of attribute edges between structure vertices and attribute vertex a_p in the transition probability $P(i)$ of G_i in the t^{th} iteration.

All α_{ip}^0s are first initialized as 1.0. We then iteratively adjust α_{ip}^{t+1} with an increment $\triangle\alpha_{ip}^t$, which denotes the weight update of attribute edges between structure vertices and attribute vertex a_p in $P(i)$. The attribute weight α_{ip}^{t+1} in the $(t+1)^{th}$ iteration is computed as

$$\alpha_{ip}^{t+1} = \frac{1}{2}(\alpha_{ip}^t + \triangle\alpha_{ip}^t) \tag{7.6}$$

To determine the extent of weight increment $\triangle\alpha_{ip}$, we design a majority vote mechanism: if a large portion of structure vertices within each cluster share the same attribute vertices with similar access counts, which means it has a good clustering tendency, then the structure weight α_{ip} should be increased; on the other hand, if structure vertices within clusters have a very random distribution on attribute or have quite diverse access counts, then the weight α_{ip} should be decreased. We define a vote measure which determines whether two structure vertices u and v have similar attributes.

$$vote_{ip}(u,v) = \begin{cases} 1, \ 1 - \dfrac{|a_{ip}(u) - a_{ip}(v)|}{|a_{ip}(u) + a_{ip}(v)|} > \epsilon \\ 0, \qquad otherwise \end{cases} \tag{7.7}$$

where $a_{ip}(u)$ specifies the count achieved by u on attribute a_p in G_i. A positive number ϵ denotes a threshold of similar extent of u and v on a_p. The weight increments $\triangle\alpha_{ip}^t$ are then calculated as

$$\triangle\alpha_{ip}^t = \frac{\sum_{j=1}^{k_i} \sum_{v \in C_j} vote_{ip}(c_j, v)}{\frac{1}{m}\sum_{q=1}^{m}\sum_{j=1}^{k_i}\sum_{v \in C_j} vote_{iq}(c_j, v)} \tag{7.8}$$

An important property of the weight self-adjustment mechanism is that the updated weights should increase the clustering objective. The detailed proof is omitted due to space limit. We will briefly illustrate this property qualitatively: if a large number of structure vertices within clusters have the similar access counts on a_p, then the weight is increased, i.e., $\alpha_{ip}^{t+1} > \alpha_{ip}^t$; on the other hand, if structure vertices within clusters have quite different access counts on a_p, the weight is then decreased, i.e., $\alpha_{ip}^{t+1} < \alpha_{ip}^t$. There must be some

Table 7.2: Trace Dataset Summary

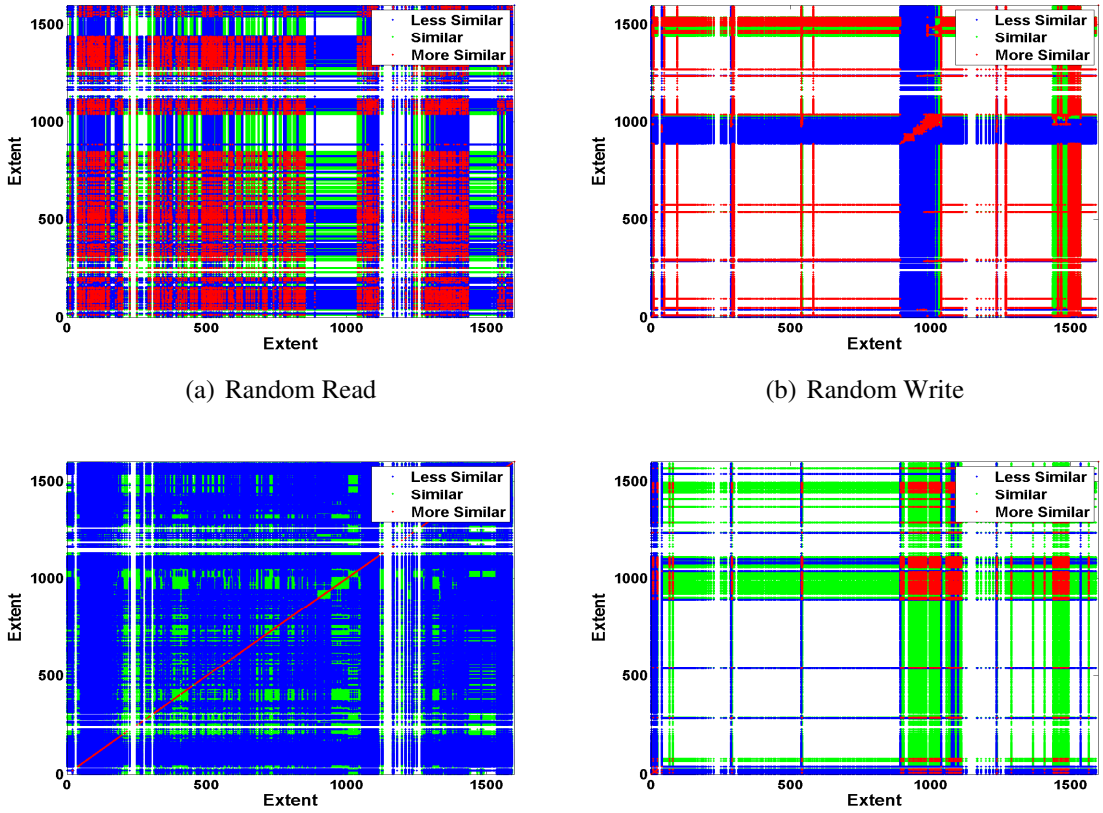
DataSet Name	Total Size (in GB)	Duration (in cycles)
Bank Trace	8,097	2,013
Store Trace	7,902	2,008
Email Trace	1,599	2,011

weights with increasing updates and some other weights with decreasing updates, since $\sum_{p=1}^{m} \alpha_{ip} = m$ is a constant. Due to some increased weights, the random walk similarities between pairwise endpoints of attribute edges with the increased weight will be further increased. As a result, these vertices tend to be clustered into the same cluster, thus increasing the clustering objective. Due to space constraint, we omit the pseudo code of our E-Cluster algorithm in this section.

7.5 Experimental Evaluation

In this section we discuss insights obtained by employing GraphLens on three different real world traces from three perspectives: spatial extent correlation analysis, temporal cycle correlation analysis and hotspot characterization. For ease of presentation, we divide similarity scores between 0 and 1 into three groups: “More Similar” (red, [0.9, 1]), “Similar” (green, (0.5, 0.9)) and “Less Similar” (blue, [0, 0.5]). In addition, the white area represents the extents without any activities in the given cycle.

We use the three storage trace datasets described in Section 7.2. The trace characteristics are summarized in Table 7.2. We build a heterogeneous trace graph for the workloads in each cycle where structure vertices represent the combinations of Lun and Extent, attribute vertices specify four access patterns of RR, RW, SR and SW. Attribute edges denote the relationships between structure vertices and attribute vertices, weighted by the corresponding access count to each data unit within the cycle. We construct 2,013, 2,008 and 2,011 trace graphs for three storage traces respectively.



(a) Random Read

(b) Random Write

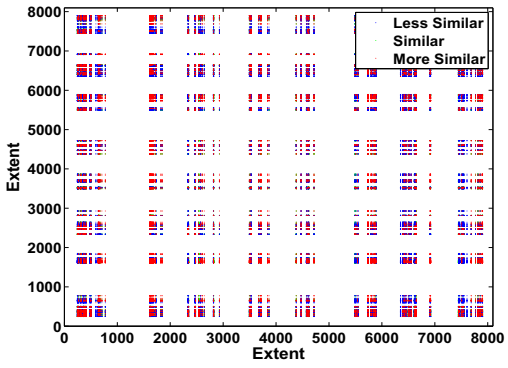
(c) Sequential Read

(d) Sequential Write

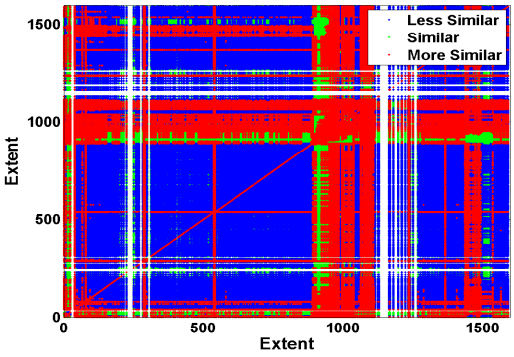
Figure 7.7: Extent Similarity on Email Trace

7.5.1 Spatial Correlation Analysis

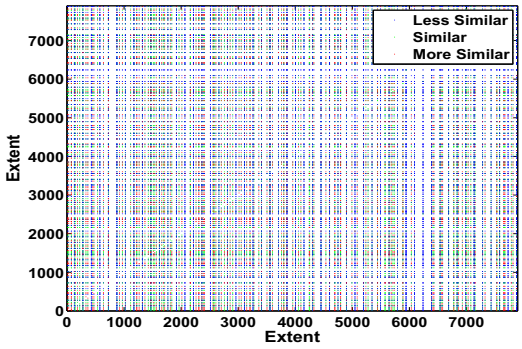
Figures 7.7 (a), (b), (c) and (d) represent the spatial similarity matrix based on each access pattern for the email trace. Figure 7.7 (a) shows that most of spatial similarity between extents based on random reads shows several regions of strong similarity. Next observe that 7.7 (b) and (c) which represents the random write and sequential read patterns are dominated by weak similarity. It is highly likely that most people usually access their emails at least once each day and mainly read the emails without any reply. Thus, most of extents are very similar based on random reads due to frequent as well as random accesses. Figure 7.7 (d) is dominated by average similarity. Sequential writes typically represent backup and replication activity in these environments and extents would be expected to be similar in behavior for this access pattern. However, since such activities are infrequent,



(a) Bank Trace



(b) Email Trace



(c) Store Trace

Figure 7.8: Unified Extent Similarity on Different Traces

the number of accesses are very low, leading to a decision of "Less Similar". This leads us to the first set of observations:

- **Observation 1:** similar behavior is exhibited both within and across volumes.
- **Observation 2:** spatial similarity varies by the dimension under consideration.
- **Observation 3:** data expresses stronger similarity under the random read access pattern.

Figures 7.8 (a), (b) and (c) exhibit the unified random walk similarity matrix on different trace datsets. We use our dynamic vote-based weight tuning method in Section 7.4.2 to learn an optimal weight assignment for four types of attribute links: RR, RW, SR and SW, to achieve high intra-cluster similarity and low inter-cluster similarity, i.e., extents within clusters have similar access patterns, while the extents in different clusters have diverse

access patterns. The similarity matrix in Figure 7.8 (a) is similar to the similarity matrix we have observed in the extent similarity case. This demonstrates that the unified random walk similarity on Bank Trace is dominated by the access pattern of random read. Due to space constraint, we omit the extent similarity on Bank trace and on Store trace.

Comparing with Figures 7.7 (b) and (d), the similarity matrix in Figure 7.8 (b) mainly depends on the access patterns of random write and sequential write. Since most of extents do not have these two kinds (random write and sequential write) of access activities extents that exhibit these activities are more alike each other and more different from extents that do not exhibit the activity. Figure 7.8 (c) shows the unified random walk similarity matrix on Store Trace, which is a relatively random distribution for each of three kinds of similarities due to the lack of clear distinctions between extent access patterns. This leads us to the following observations.

- **Observation 4:** when all data exhibits all types of access pattern, the strongest access pattern (which is most often random read) dominates the similarity metric. This implies that data placement taking only random read patterns into consideration is likely to provide good results.
- **Observation 5:** when access type distributions are not uniform across all extents, extents that exhibit more rare access patterns have stronger similarity under a unified metric. This implies that under such circumstances, data placement must first consider the unified metric to identify broader distinctions between extents and then consider random reads as a secondary metric.
- **Observation 6:** when unified extent similarity weighted on all access patterns exhibit a relatively random distribution as shown in Figure 7.8 (c), this indicates that there is no need to further explore attribute-specific spatial access patterns.

7.5.2 Hotspot Characterization

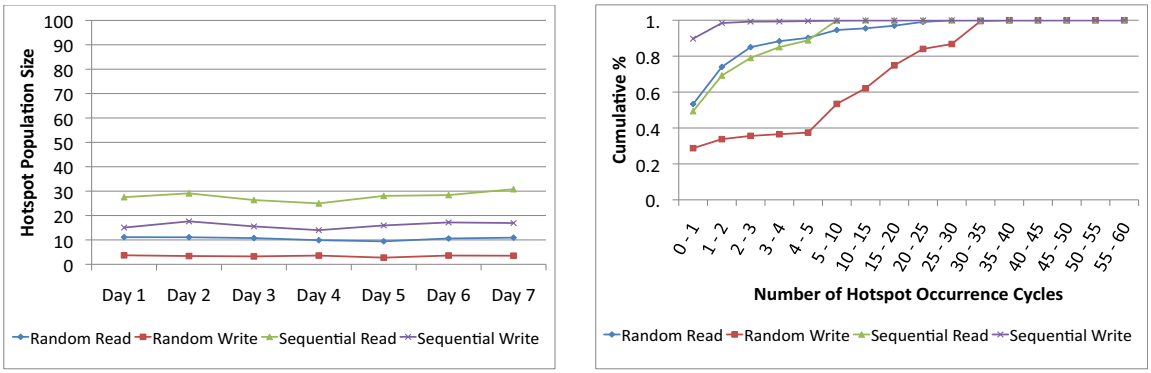
Hotspots can be defined as regions that have relatively higher activity (hence "temperature" or "heat") in comparison to its surroundings. Understanding hotspot characteristics is essential to data placement strategies, such as caching at host or storage server by utilizing the "recency" [153, 142], and tiering by exploring the "frequency" aspect [154, 143]. By using GraphLens, extents are classified into "Hot", "Warm" and "Cold" clusters for each cycle. An extent that appears in the hot cluster at time t is referred to as a hotspot at time t. A single extent can exhibit hotspot behavior in multiple cycles and multiple extents can exhibit hotspot behavior in the same cycle. Our dynamic weight assignment and update at each clustering iteration reduce the possible bias introduced by a single attribute dominating the clustering outcome. We use the following measures to classify hotspots from the temporal and spatial clustering analysis results:

- **Population Size:** a summary measure which describes the number of unique extents that exhibit hotspot behavior within a window (24 hours in this study) of observation i.e. size of the hotspot.
- **Intra-window stability or Burstiness:** frequency distribution of number of hotspot occurrences for a each unique extent *within a window of observation*. This measure is indicative of the burstiness and durability of hotspot behavior within each time window.

Due to space constraint, we omit the hotspot characterization of Email Trace.

Banking Transactions Workload

Figure 7.9(a) shows the variation in hotspot population size over 7 days for four different access patterns. The x-axis and the y-axis represent the day and the percentage of total dataset population that exhibited hotspot behavior at any time during the day for a specific access pattern. We see that the hotspot population size remains fairly stable from



(a) Bank Trace: Population Size

(b) Bank Trace: Burstiness

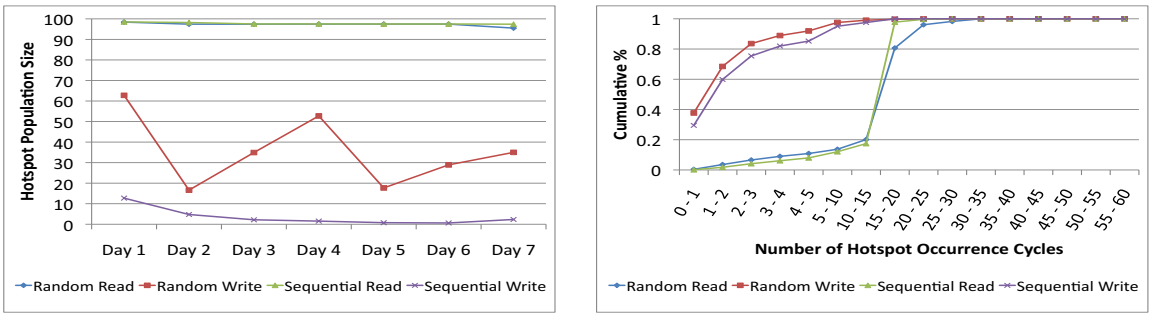
Figure 7.9: Bank Trace

day to day for all workloads. Random Read (3%) and Random write (10%) are most stable. Sequential Read (30%) and Sequential Write (17%) activities span relatively larger population sizes but remain small compared to the total dataset size.

Figure 7.9(b) shows the frequency distribution during a 24 hour period for which an extent exhibited hotspot behavior. Random write workload exhibits the least burstiness with nearly 63% of the hotspots lasting longer than 75 minutes. Random read and sequential read hotspots are relatively more bursty with only 10% of the sequential read and random read hotspots lasting longer than 75 minutes. On the other hand, sequential write is the most bursty with 90% of hotspots lasting less than 15 minutes in a day and nearly all hotspots lasting less than 30 minutes. We conjecture that random write workloads for this application are probably best serviced by a tiering strategy. On the other hand, random reads and sequential reads contain a mix of bursty and stable hotspot behavior, a combination of caching (to catch bursty hotspots) and tiering (to catch more long term behavior) could be used. Sequential writes exhibit highly bursty behavior, which could be addressed with prefetch caching.

Store Backend Workload

In the Store trace (Figure 7.10 (a)), almost all data exhibits hotspot behavior at some point during the day. Sequential write and random write hotspots are limited to a smaller fraction of the dataset (3% and 35% respectively). However sequential write and random



(a) Store Trace: Population Size

(b) Store Trace: Burstiness

Figure 7.10: Store Trace

write hotspots show large variations in population size over the week.

Figure 7.10 (b) shows the frequency distribution during a 24 hour period for which an extent exhibited hotspot behavior. Sequential reads and random read patterns show very identical behavior with 80% of the extents exhibit hot spot behavior for nearly 4 to 5 hours a day. In comparison, Sequential writes and random writes are relatively bursty with nearly 60% and 70% respectively of the hotspots exhibiting hotspot behavior for less than 30 minutes in a day. Given the large population size and the low burstiness, these access patterns may be effectively addressed by provisioning a high performance tier (assuming that 8TB of cache may not be viable option at every host and population such a large cache may itself take several hours).

7.6 Conclusions

We have presented a novel graph analytics framework, GraphLens, for mining and analyzing real storage traces. We model storage traces as heterogeneous trace graphs to incorporate multiple complex and heterogeneous factors into a unified analytic framework. An innovative graph clustering method is proposed to identify and discover spatial correlations and hotspot characterization. We design an dynamic weight tuning method to combine multiple correlations into a unified similarity measure with optimal weights.

CHAPTER 8

SERVICECLUSTER: CLUSTERING SERVICE NETWORKS WITH ENTITY, ATTRIBUTE AND LINK HETEROGENEITY

Many popular web service networks are content-rich in terms of heterogeneous types of entities and links, associated with incomplete attributes. Clustering such heterogeneous service networks demands new clustering techniques that can handle two heterogeneity challenges: (1) multiple types of entities co-exist in the same service network with multiple attributes, and (2) links between entities have diverse types and carry different semantics. Existing heterogeneous graph clustering techniques tend to pick initial centroids uniformly at random, specify the number k of clusters in advance, and fix k during the clustering process. In this chapter, we present ServiceCluster, a novel heterogeneous Service network Clustering algorithm with four unique features. First, we incorporate various types of entity, attribute and link information into a unified distance measure. Second, we design a Discrete Steepest Descent method to naturally produce initial k and initial centroids simultaneously. Third, we propose a dynamic learning method to automatically adjust the link weights towards clustering convergence. Fourth, we develop an effective optimization strategy to identify new suitable k and k well-chosen centroids at each clustering iteration. Extensive evaluation on real datasets demonstrates that ServiceCluster outperforms existing representative methods in terms of both effectiveness and efficiency.

8.1 Introduction

Efficient web service analysis is widely recognized as an interesting and challenging research problem, which has received heated attention recently [122, 126, 127, 129, 105, 155, 124, 123, 125, 128, 156, 157, 158]. As more and more people are engaged in service network analysis, we witness many forms of heterogeneous service networks in which entities

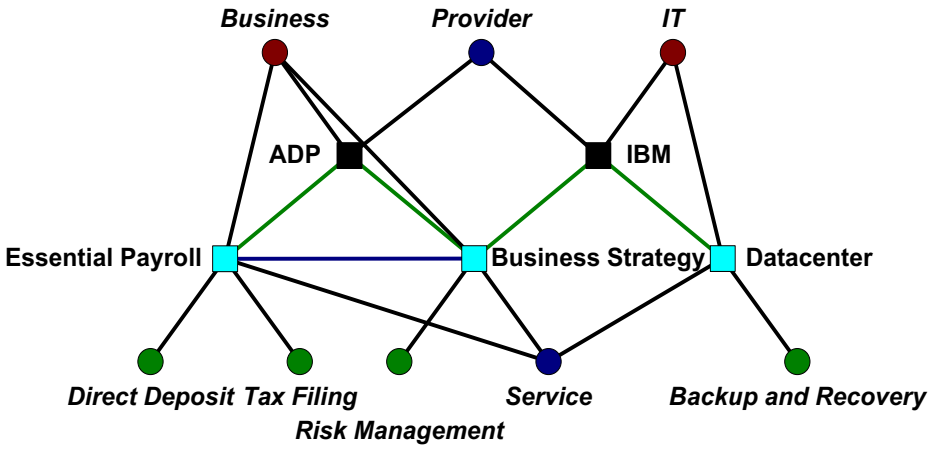


Figure 8.1: A Heterogeneous Service Network from IBM Knowledge Base

are of different types and are interconnected through heterogeneous types of links, representing different kinds of semantic relations. Figure 8.1 presents a real service network from IBM knowledge base. There are two kinds of object vertices: black "Provider" nodes and cyan "Service" vertices. Each "Provider" vertex may have two kinds of attributes: blue "Type" attribute and ochre "Category" property. Each "Service" vertex may contain three types of properties: blue, ochre and green attribute vertices specify service's "Type", "Category" and "Capability", respectively. On the other hand, there are three kinds of links: a green edge represents the "Provides" relationship between "Provider" and "Service"; a blue line specifies the structure relationship between objects with the same type; a black edge denotes the attribute edge between object and its attribute.

Analyzing and mining such heterogeneous service networks can provide new insights about how entities influence and interact with each other and how ideas and opinions propagate on service networks. For example, clustering online service network may help understanding consumer segmentation for service marketing. Clustering heterogeneous social network becomes an interesting and challenging research problem which has received much attention recently [19, 20, 28, 21, 23, 63, 51]. However, clustering heterogeneous networks with multiple types of entities, attributes and links poses a number of new challenges.

- Different types of entities co-exist in the same information network, and each type of en-

tities may have diverse types of links and different sets of attributes. It is very challenging to decide the importance of various types of entities, attributes and links to improve the clustering quality. In Figure 8.1, if we want to partition "Provider"s into clusters based on the structure links between "Provider"s and their attribute information, then two associated attributes of "Type" and "Category" may have different degrees of importance. A weight learning method for different types of links in terms of their contribution in the clustering is a possible solution.

- Most of existing heterogeneous graph clustering methods [19, 20, 28, 21, 23, 63, 51] require the number of clusters to be specified in advance. It is hard to decide for inexperienced users. The usual method is to compare the results of multiple runs with different k values and choose the best one in terms of a given criterion. However, the repeated clustering run could be expensive for large datasets.
- The choice of initial cluster centroids have a great effect on the clustering result. Existing studies usually choose centroids uniformly at random from data points, which makes it difficult to find high quality clustering results.
- None of existing literatures [20, 28, 21, 23, 51] on weight learning methods has studied the impact of dynamic weight assignment on the dataset. For example, the original similarity (or distance) scores may need to be updated due to iterative update of link weights. In addition, it is necessary to recalculate the better k and the better centroids with the update on similarity (or distance) scores, which actually change the shape and scale of dataset.

With these new challenges in mind, in this chapter we develop an innovative dynamic clustering approach of heterogeneous service networks, called ServiceCluster. Our approach makes a number of original contributions.

- We propose a unified random walk distance measure integrating various types of entities, attributes and links to measure vertex closeness on a heterogeneous network.
- We design a Discrete Steepest Descent method to naturally determine the number of clus-

ters and generate the corresponding centroids in a heterogeneous network.

- We propose a dynamic learning method to automatically adjust the link weights towards the direction of clustering convergence.
- We argue that the correct choice of k often depends on the shape and scale of dataset. Thus we develop an effective optimization strategy to identify new suitable k and k well-chosen centroids upon the update on similarity scores and weight tuning to continuously improve the clustering quality at each clustering iteration.

8.2 Related Work

Web service discovery and management has been a heated topic in recent years [122, 126, 127, 129, 105, 155]. Skoutas et al. [124] proposed a methodology for ranking and clustering the relevant web services based on the notion of dominance, which apply multiple matching criteria without aggregating the match scores of individual service parameters. Xiao et al. [123] proposed a context modeling approach which can dynamically handle various context types and values. Almulla et al. [125] presented a web services selection model based on fuzzy logic and proposed a fuzzy ranking algorithm based on the dependencies between proposed quality attributes. Liu et al. [128] proposed a heuristic social context-Aware trust network discovery algorithm, H-SCAN, by adopting the K-Best-First Search (KBFS) method and some optimization strategies. Kumara et al. [157] proposed a hybrid web-service clustering approach with considering both ontology learning and IR-based term similarity.

Graph clustering and graph classification has attracted active research in the last decade [19, 13, 14, 20, 28, 9, 21, 23, 63, 51, 65]. Shiga et al. [19] presented a clustering method which integrates numerical vectors with modularity into a spectral relaxation problem. S-CAN [13] is a structural clustering algorithm to detect clusters, hubs and outliers in networks. SA-Cluster [20], Inc-Cluster [28] and BAGC [21] perform clustering based on both structural and attribute information by incorporating attributes into an attributed graph.

RankClass [9] groups objects into pre-specified classes, while generating the ranking information for each type of object. GenClus [23] proposed a model-based clustering method for heterogeneous graphs with different link types and attribute types.

Some recent studies have shown the clustering quality can be enhanced by selecting a good k or by choosing good initial centroids. K-Means++ [140] can find a clustering that is $O(logk)$-competitive to the optimal K-Means solution by specifying a procedure to initialize the cluster centers before proceeding with the K-Means iterations. G-Means [159] runs K-Means with increasing k in a hierarchical fashion until the data assigned to each K-Means center are Gaussian.

To our knowledge, this work is the first one to address the problem of clustering heterogeneous service networks by simultaneously refining the link weights, the k value and the cluster centroids to progressively enhance the clustering quality in each clustering iteration.

8.3 Problem Statement

A **heterogeneous service network** is denoted as $G = (V, A, E)$, where $V = \bigcup_{i=1}^{N} V_i$ represents the set of N types of entity vertices, such as services, providers and customers, $A = \bigcup_{i=1}^{M} A_i$ denotes the set of M kinds of associated attribute vertices, $E = \bigcup_{i=1}^{L} E_i$ specifies the set of L types of edges among entity vertices and attribute vertices. An attribute vertex $v \in A_i$ denotes some concrete value for the i^{th} kind of attributes. For example, an attribute of provider "Category" has a value of "IT". An edge may exist between entities with the same type (e.g., "Cloud Computing" and "Datacenter" are similar), or between entities with different types (e.g., "Amazon" provides a service of "Datacenter"), or between entities and attributes (e.g., "IBM"'s "Category" is "IT"). We denote the number of each type of entity vertices, attribute vertices or links as $n_i = |V_i|$ $(1 \leq i \leq N)$, $m_j = |A_j|$ $(1 \leq j \leq M)$, $l_k = |E_k|$ $(1 \leq k \leq L)$ respectively. The total number of entities, attributes or links is equal to $n = \sum_{i=1}^{N} n_i$, $m = \sum_{j=1}^{M} m_j$, $l = \sum_{k=1}^{L} l_k$, respectively.

Given a heterogeneous service network G, the problem of Heterogeneous **Service Net-**

work Clustering (ServiceCluster) is to partition the objective entities V_i with the i^{th} type into k_i disjoint clusters $C_1, C_2, \ldots, C_{k_i}$, where $i \in \{1, 2, \ldots, N\}$, $V_i = \bigcup_{p=1}^{k_i} C_p$ and $C_p \bigcap C_q = \phi$ for $\forall p, q, 1 \leq p, q \leq k_i, p \neq q$, to ensure: (1) the clustering of the objective entities has an optimal number of clusters in terms of the shape and scale of dataset; (2) the entities within each cluster are densely connected, while the entities in different clusters are distant from each other; (3) the entities within each cluster have similar interactions with other types of entities, while the entities in different clusters have diverse interplays with other types of entities; and (4) the entities within clusters may have similar properties, while the entities in different clusters have diverse attribute values.

8.4 A Unified Weighted Distance Measure

In a heterogeneous network with various types of entities, attributes and links, each entity is associated with a set of multiple types of entities through the links between entity vertices, and a set of different kinds of attributes through the connections between entity vertices and attribute vertices, we propose to use a unified distance measure based on the neighborhood random walk model to integrate various types of link information. In the heterogeneous network, there exists a random walk path between two entities $v_1, v_2 \in V_i$ if (1) v_1 and v_2 have the same peer entity $v_3 \in V_i$; (2) both v_1 and v_2 are connected to the same entity $v_4 \in V_j$ with different type; or (3) v_1 and v_2 have the same attribute value $v_5 \in A_k$. If there are multiple paths connecting v_1 and v_2, then they are close. On the other hand, if there are very few or no paths between v_1 and v_2, then they are far apart.

Definition 25 (Transition Probability) *Let $V = \bigcup_{i=1}^{N} V_i$ be the set of N types of entities, $A = \bigcup_{i=1}^{M} A_i$ be the set of M kinds of associated attributes, the weighted transition matrix T*

of a heterogeneous network G is defined below.

$$P = \left[\begin{array}{ccc:ccc} P_{11} & \dots & P_{1N} & P_{1(N+1)} & \dots & P_{1(N+M)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ P_{N1} & \dots & P_{NN} & P_{N(N+1)} & \dots & P_{N(N+M)} \\ \hdashline P_{(N+1)1} & \dots & P_{(N+1)N} & P_{(N+1)(N+1)} & \dots & P_{(N+1)(N+M)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ P_{(N+M)1} & \dots & P_{(N+M)N} & P_{(N+M)(N+1)} & \dots & P_{(N+M)(N+M)} \end{array}\right] \tag{8.1}$$

$$T = \omega \cdot P$$

where each P_{jk} in P represents the transition probability for some kind of links. Each ω_{jk} in weight matrix $\omega = [\omega_{11}, \dots, \omega_{1(N+M)}; \dots; \omega_{(N+M)1}, \dots, \omega_{(N+M)(N+M)}]$ sp-ecifies the weight of P_{jk}. According to the types of sources and destinations, P is divided into four parts: (1) P_{jk} $(1 \le j,k \le N)$ is a $n_j \times n_k$ block matrix representing the transition probability between entity vertices. Each entry in P_{jk} is the original edge value between entities, e.g., the similar degree between two entity vertices with the same type of "Service", or denoting whether $v \in V_k$ is provided by $u \in V_j$ when u is a "Provider" and v is a "Service"; (2) a $n_j \times m_{k-N}$ block matrix P_{jk} $(1 \le j \le N, N+1 \le k \le N+M)$ specifies the transition probability from entities to attributes. Each element in P_{jk} has a binary value of 0 or 1 specifying whether the entity holds the attribute value, e.g., "IBM" has a "Category" of "IT"; (3) P_{jk} $(N+1 \le j \le N+M, 1 \le k \le N)$ is a $m_{j-N} \times n_k$ block matrix denoting the transition probability from attributes to entities. Each entry specifies whether the attribute value is owned by the entity, e.g., a "Category" of "IT" is possessed by "IBM"; and (4) P_{jk} $(N+1 \le j,k \le N+M)$ is a $m_{j-N} \times m_{k-N}$ block matrix with all 0s since there is no connection between attributes.

We argue that each type of links may have different degrees of contribution in the clustering process. Thus we assign an individual weight ω_{ij} for each kind of transition prob-

abilities to generate the weighted version T of P. Notice that ω_{ij} may not be equal to ω_{ji} since the information flow between two different types of vertices may be bidirectional. In our current experiment setup, all weights are initialized as 1.0*. Since each row of transition matrix should sum to* 1*, we further perform the row-wise normalization for T.*

Based on the definition of transition probability, we actually split the original transition operation into two steps: (1) inspect the weight of source's neighbors and choose some kind of vertices with the largest weight; and (2) check the original edge value between source and vertices with the largest weight and choose some vertex with the largest edge value as destination.

Definition 26 (Unified Neighborhood Random Walk Distance) *Let T be the weighted transition probability of a heterogeneous network G, l be the length that a random walk can go, and $c \in (0, 1)$ be the restart probability, the unified random walk distance $d(u, v)$ from $u \in V$ to $v \in V$ in G is defined as follow.*

$$d(u, v) = \sum_{\substack{\tau: u \rightsquigarrow v \\ length(\tau) \leq l}} p(\tau) c (1 - c)^{length(\tau)} \tag{8.2}$$

where τ is a path from u to v whose length is $length(\tau)$ with transition probability $p(\tau)$. $d(u, v)$ reflects the vertex closeness based on multiple types of link information.

The matrix form of the unified distance is given as follow.

$$R = \sum_{\gamma=1}^{l} c(1 - c)^{\gamma} T^{\gamma} \tag{8.3}$$

8.5 Heterogeneous Service Network Clustering

With the unified random walk distance as an input, ServiceCluster first identifies initial k_i and initial centroids by using a Discrete Steepest Descent method. At each inner iteration, it follows the K-Medoids clustering method [25]: assign vertices to their closest centroids and

select the most centrally located point in a cluster as new centroid. At each outer iteration, the optimal weights are generated by maximizing the clustering objective. It further splits or merges current clusters to identify new k_i and new centroids. This process is repeated until convergence.

8.5.1 Selection of k_i and Initial Centroids

We will address two main issues in the initialization step: (1) initial k_i setup and (2) cluster centroid initialization.

We argue that choosing k_i randomly without prior knowledge by existing heterogeneous clustering methods often leads to incorrect clustering results. In addition, good initial centroids are essential for the success of partitioning-based clustering algorithms. We propose a Discrete Steepest Descent method (DSD) to provide a natural way to determine the number of clusters and the initial centroids simultaneously. Intuitively, if we choose a local densest vertex, which is similar to the most peer vertices, in its neighborhood as centroids, then this will maximize the within-cluster similarity on the group, which consists of the centroid and its neighbors. To find such local densest vertices, we first define the density $D_{V_i}(v)$ of an entity $v \in V_i$ on V_i in terms of the unified distance measure below.

$$D_{V_i}(v) = \sum_{u \in V_i, u \in \epsilon-neighborhood\ of\ v, \epsilon \in \mathbb{Z}_+} d(u, v) \tag{8.4}$$

where $d(u, v)$ is the unified random walk distance from u to v in G. $D_{V_i}(v)$ summarizes the similarities between v and its *ϵ-neighborhood* in V_i. According to the definition of clustering objective in Eq.(8.9), $D_{V_i}(v)$ is equivalent to the objective of cluster which takes v as centroid and consists of v and its *ϵ-neighborhood* in V_i.

The Steepest Descent method (SD) is an effective first-order optimization algorithm to find a local maximum (or minimum) of a function. The local maximization version of SD starts with an initial point x_0, and iteratively executes the following steps to update the current iterate x_{t-1} by a step γ_t from x_0 in the gradient direction which increases the

objective $f(x)$ until moving to a critical point, i.e., $x_t = x_{t-1}$, which is hopefully the desired local maximum.

$$\begin{aligned} \gamma_t &= argmax_{\gamma \geq 0}\ f(x_{t-1} + \gamma f'(x_{t-1})) \\ x_t &= x_{t-1} + \gamma_t f'(x_{t-1}) \end{aligned} \tag{8.5}$$

where γ_t is the steepest step to increase $f(x)$ at the fastest rate and therefore make the biggest change.

So far most existing first order optimization methods such as Steepest Descent or second order optimization models such as Newton-Raphson have assumed a continuous differentiable search space. However, the search space in a graph is not continuous but discrete, i.e., made up of individual vertices. Thus, we propose the DSD method to find the local maxima of $D_{V_i}(v)$ in the heterogeneous network G.

$$\begin{aligned} \gamma_t = argmax_{\gamma: v_{t-1} \to v_t, v_t \in \epsilon-neighborhood\ of\ v_{t-1}, \epsilon \in \mathbb{Z}_+} D_{V_i}(v_t) \\ v_{t-1} \xrightarrow{\gamma_t} v_t \end{aligned} \tag{8.6}$$

where the gradient $D'_{V_i}(v_{t-1})$ is approximated by $D_{V_i}(v_t) - D_{V_i}(v_{t-1})$ and the steepest step is directed to the densest vertex in the *ϵ-neighborhood* of v_{t-1}. The process terminates when $D_{V_i}(v_t) \leq D_{V_i}(v_{t-1})$. This approach is typically much faster than an exhaustive search when V_i is large.

Based on the DSD method, the initialization of k_i and centroids for the objective entities V_i is presented in Algorithm 10. Each iteration in the DSD process forms a path of steepest descent from a starting vertex v_0 to a local maximum v_{t-1} of density. When the DSD procedure carries out a multi-dimensional search from an unvisited vertex v_{t-1} to a visited vertex v_t and $D_{V_i}(v_t) > D_{V_i}(v_{t-1})$, we incorporate the current path from v_0 to v_{t-1} and the previous path including v_t into a tree of steepest descent since $v_0, \ldots, v_t$ have the same local

Algorithm 10 Initialization of k_i and Centroids

Input: a service network G=(V, A, E), the objective entities V_i, the random walk distance R, a set of unvisited entities S, a set of visited entities T.

Output: k_i, centroids $c_1, ..., c_{k_i}$.

1: S=V_i, T=ϕ and k_i=0;
2: **while** $S \neq \phi$
3: Choose one starting vertex v_0 randomly from S;
4: **while** $D_{V_i}(v_{t-1})$<$D_{V_i}(v_t)$ by DSD
5: **if** $v_t \in T$ and $v_t \in C_k (k \in \{1, \ldots, k_i\})$
6: S=S−$\{v_0, \ldots, v_{t-1}\}$ and T=T+$\{v_0, \ldots, v_{t-1}\}$;
7: C_k=C_k+$\{v_0, \ldots, v_{t-1}\}$ and **goto** Step 2;
8: S=S−$\{v_0, \ldots, v_{t-1}\}$ and T=T+$\{v_0, \ldots, v_{t-1}\}$;
9: k_i=k_i+1, c_{k_i}=v_{t-1} and C_{k_i}=$\{v_0, \ldots, v_{t-1}\}$;
10: Return k_i, $c_1, ..., c_{k_i}$ and $C_1, ..., C_{k_i}$.

maximum. The DSD process is repeated until each vertex is assigned to one and only DSD path or tree. Thus, we present a quite natural way to generate k_i centroids (local maxima) and k_i clusters (paths or trees).

When the heterogeneous network is dense enough, or we run enough random walk propagations on the heterogeneous network, the DSD method exhibits superior performance on the selection of k_i. Assuming that each entity is at least in the *ϵ-neighborhood* of one entity vertex with local maximum of density, we can generate the following theoretical property.

Theorem 24 *The lower bound of the optimal number of clusters is k_i by DSD.*

Proof. Let C_1 and C_2 be two arbitrary clusters by DSD, c_1 and c_2 be the corresponding centroids respectively. We try to prove the clustering objective will be reduced if we combine C_1 and C_2 into one cluster, i.e, decrease k_i. There are two possible cases to be discussed separately: (1) if the centroid in the new combined cluster is arbitrary one of c_1 and c_2, say c_1, in terms of the definition of clustering objective in Eq.(8.9), then the cluster objective on $\forall v \in C_2$ will be reduced due to $\sum_{v \in C_2} d(v, c_1) \leq \sum_{v \in C_2} d(v, c_2)$; and (2) if the centroid c in the combined cluster is neither c_1 nor c_2, then the cluster objective on both $\forall u \in C_1$ and $\forall v \in C_2$ will be reduced due to $\sum_{u \in C_1} d(u, c) \leq \sum_{u \in C_1} d(u, c_1)$ and $\sum_{v \in C_2} d(v, c) \leq \sum_{v \in C_2} d(v, c_2)$.

8.5.2 Vertex Assignment and Centroid Update

Although the DSD method can produce a good clustering when facing a dense heterogeneous network, we may need to further refine clusters when many entity vertices locate outside the *ϵ-neighborhood* of any local maximum since each DSD step adopts a local optimization strategy in the range of *ϵ-neighborhood* of current iterate v_{t-1}. For example, there is a DSD path $v_0 \rightarrow v_1 \rightarrow v_2$ where v_0 is the starting point and v_2 is the local maximum. Although v_1 may be very similar to both v_0 and v_2, the similarity between v_0 and v_2 may be quite small.

For the objective entities V_i, with k_i centroids in the t^{th} iteration, we assign each vertex $u \in V_i$ to its closest centroid $c^* = argmax_{c_j^t \in \{c_1^t, \ldots, c_{k_i}^t\}} d(u, c_j^t)$. When all vertices are assigned to some cluster, the centroid will be updated with the most centrally located vertex in each cluster. To find such a vertex, we first compute the "average point" $\overline{u}$ of a cluster C_j in terms of the unified distance measure as

$$d(\overline{u}, v) = \frac{1}{|C_j|} \sum_{w \in C_j} d(w, v), \forall v \in V \tag{8.7}$$

Thus $d(\overline{u}, :)$ is the average unified distance vector for cluster C_j. Then we find the new centroid c_j^{t+1} in C_j as

$$c_j^{t+1} = argmin_{v \in C_j} ||d(\overline{v}, :) - d(\overline{u}, :)|| \tag{8.8}$$

Therefore we find the new centroid c_j^{t+1} in the $(t+1)^{th}$ iteration whose unified random walk distance vector is the closest to the cluster average.

8.5.3 Objective Function

The objective of clustering is to maximize within-cluster similarity between centroids and member vertices.

Definition 27 *[Graph Clustering Objective Function] Let $G = (V, A, E)$ be a heterogeneous network with N types of entity vertices and M kinds of associated attribute vertices, $\omega_{jk}(1 \leq j, k \leq N + M)$ be the weight of each kind of links, and k_i be the number of clusters for the objective entities V_i. The goal of the heterogeneous network clustering is to find k_i partitions $\{C_p\}_{p=1}^{k_i}$ such that $V_i = \bigcup_{p=1}^{k_i} C_p$ and $C_p \bigcap C_q = \phi$ for $\forall p, q, 1 \leq p, q \leq k_i, p \neq q$, and the following objective function $O(\{C_p\}_{p=1}^{k_i}, \omega)$ is maximized.*

$$O(\{C_p\}_{p=1}^{k_i}, \omega) = \sum_{p=1}^{k_i} \sum_{v \in C_p} d(v, c_p)$$
$$\omega = [\omega_{11}; \ldots; \omega_{1(N+M)}; \ldots; \omega_{(N+M)1}; \ldots; \omega_{(N+M)(N+M)}] \tag{8.9}$$

s.t. $\omega_{jk} \geq 0, \sum_{j=1}^{N+M} \sum_{k=1}^{N+M} \omega_{jk} = (N + M)^2$.

The original objective is a polynomial function of ω with non-negative coefficients. We reformulate it as follow.

$$O(\{C_p\}_{p=1}^{k_i}, \omega) = \sum_{m=1}^{n} a_m \prod_{j=1,k=1}^{N+M} (\omega_{jk})^{p_{mjk}}$$
$$a_m \geq 0, p_{mjk} \geq 0, p_{mjk} \in \mathbb{Z}, \sum_{j=1,k=1}^{N+M} p_{mjk} \leq l \tag{8.10}$$

s.t. $\omega_{jk} \geq 0, \sum_{j=1}^{N+M} \sum_{k=1}^{N+M} \omega_{jk} \leq (N + M)^2$.

where the objective consists of n within-cluster polynomial terms, a_m is the coefficient of the m^{th} term, p_{mjk} is the exponent of ω_{jk} in the m^{th} term, and l is the length that a random walk can go. Notice that we replace the constraints in Eq.(8.9) with those in Eq.(8.10) since $O(\{C_p\}_{p=1}^{k_i}, \omega)$ is monotonically increasing with non-negative ω such that it can achieve the same maximum on both constraint spaces.

For ease of presentation, we substitute ω with μ such that their components are in one-

to-one correspondence.

$$
\begin{aligned}
O(\{C_p\}_{p=1}^{k_i}, \mu) = \sum_{m=1}^{n} a_m \prod_{j=1}^{(N+M)^2} \mu_j^{p_{mj}}, a_m \geq 0, p_{mj} \geq 0, \\
p_{mj} \in \mathbb{Z}, \sum_{j=1}^{(N+M)^2} p_{mj} \leq l, \mu = [\mu_1; \ldots; \mu_{(N+M)^2}]
\end{aligned}
\tag{8.11}
$$

s.t. $\mu_j \geq 0, \sum_{j=1}^{(N+M)^2} \mu_j \leq (N+M)^2$.

where p_{mj} *is the exponent of* μ_j *in the* m^{th} *term.*

8.5.4 Weight Optimization

The original optimization problem is a high-dimensional polynomial programming problem. On the other hand, the polynomial objective contains massive variables such that we can not directly solve the KKT system of polynomial equations. It is very hard to perform function trend identification and estimation to determine the convexity of the optimization problem. Thus, there may exist no verifiable sufficient conditions for global optimality. We convert the optimization problem in Eq.(8.11) to the following equivalent problem by setting $\mu_j = e^{\nu_j}$.

$$
\begin{aligned}
O(\{C_p\}_{p=1}^{k_i}, \nu) = \sum_{m=1}^{n} a_m e^{\nu^T p_m}, \nu = [\nu_1; \ldots; \nu_{(N+M)^2}], a_m \geq 0, \\
p_{mj} \geq 0, p_{mj} \in \mathbb{Z}, \sum_{j=1}^{(N+M)^2} p_{mj} \leq l, p_m = [p_{m1}; \ldots; p_{m((N+M)^2)}]
\end{aligned}
\tag{8.12}
$$

s.t. $\sum_{j=1}^{(N+M)^2} e^{\nu_j} \leq (N+M)^2$.

Notice that $O(\{C_p\}_{p=1}^{k_i}, \nu)$ is convex since it is a conic combination of convex functions $e^{\nu^T p_m}$. $\Theta = \{\nu | \sum_{j=1}^{(N+M)^2} e^{\nu_j} \leq (N+M)^2\}$ is convex since it has a less-than constraint and $\sum_{j=1}^{(N+M)^2} e^{\nu_j}$ is convex. We utilize the successive convex approximation method (SCA) [113]

Algorithm 11 Splitting and Merging of Clusters

Input: a service network G=(V, A, E), the objective entities V_i, the random walk distance R, k_i clusters $C_1, ..., C_{k_i}$.

Output: k_i, centroids $c_1, ..., c_{k_i}$.

1: Calculate $D_{C_j}(v)$ for each vertex v, $\forall v \in C_j$, in each cluster C_j;
2: find all local maxima $c_{j1}, ..., c_{jk_j}$ in each cluster C_j by Algorithm 10;
3: Split each cluster C_j into subclusters $C_{j1}, ..., C_{jk_j}$;
4: **for** each local maximum c
5: Compute $D_{V_i}(c)$ and $D_{V_i}(v)$, $\forall v \in \epsilon$-*neighborhood* of c;
6: **if** $D_{V_i}(v) > D_{V_i}(c)$, $v = argmax_{u \in \epsilon-neighborhood\ of\ c} D_{V_i}(u)$
7: Merge C_{ps} and C_{qt} where $v \in C_{ps}$, $c \in C_{qt}$, $C_{ps} \neq C_{qt}$;
8: Update k_i;
9: Return k_i, $c_1, ..., c_{k_i}$ and $C_1, ..., C_{k_i}$.

to maximize the convex objective on the convex set.

8.5.5 Splitting and Merging of Clusters

Due to the weight adjustment after each clustering iteration, we need to recalculate the unified random walk distance. This update operation essentially changes the shape and scale of dataset. We argue that a fixed k_i is no longer applicable to the dataset with changed shape. Thus we propose a dynamic adjustment method of k_i to identify a new suitable k_i to keep improving the clustering quality. Different from hierarchy-based clustering, our method may make k_i increase, decrease, or keep unchanged by splitting and merging current clusters after each iteration.

The cluster adjustment algorithm is presented in Algorithm 11. Instead of rediscovering local maxima of density based on the entire graph, we update local maxima with the prior knowledge of existing clustering result to continuously enhancing the clustering quality. It first calculates the density of each vertex on own cluster and find new potential local maxima in each cluster. The algorithm then splits each cluster into subclusters based on new DSD trees or paths. It figures out the density of local maxima and their ϵ-*neighborhood* on the entire graph. The density of vertices in the ϵ-*neighborhood* of a local maximum may be larger than the density of the local maximum due to the distance update. If the densest vertex in the ϵ-*neighborhood* has a larger density than the local maximum, then we merge

Algorithm 12 Heterogeneous Service Network Clustering

Input: a service network G=(V, A, E), the objective entities V_i, a length limit l of random walk paths, a restart probability c.
Output: k_i clusters $C_1, ..., C_{k_i}$.

1: ω=μ=$\mathbf{1}$;
2: Calculate T and R;
3: k_i and initial centroids $c_1, ..., c_{k_i}$ by Algorithm 10;
4: Repeat until the weight vector μ converges:
5: Repeat until the objective $O(\{C_p\}_{p=1}^{k_i}, \mu)$ converges:
6: Assign each vertex v to c^*=$argmax_{c_j} d(v, c_j)$;
7: Update c_j=$argmin_{v \in C_j} \|d(\overline{v}, :) - d(\overline{u}, :)\|$;
8: Run the SCA method to solve $(O(\{C_p\}_{p=1}^{k_i}, \mu))$ to produce ω;
9: Re-calculate T and R with the optimal ω;
10: Update k_i and $c_1, ..., c_{k_i}$ by Algorithm 11;
11: Return k_i clusters $C_1, ..., C_{k_i}$.

two subclusters, where the densest vertex and the local maximum stay in, into a new cluster until all subclusters are scanned.

8.5.6 Clustering Algorithm

By assembling different parts, our heterogeneous network partitioning algorithm is presented in Algorithm 12. ServiceCluster consists of five main tasks: (1) initialization of k_i, (2) vertex assignment, (3) centroid update, (4) weight optimization, and (5) cluster adjustment and update of k_i, each with the goal of maximizing the clustering objective. Tasks (2)-(3) are common to partitioning clustering algorithms. The other three tasks are the novelty of this work.

8.6 Experimental Evaluation

We have performed extensive experiments to evaluate the performance of ServiceCluster on real graph datasets.

8.6.1 Experimental Datasets

We modify the BSBM data generator [139] and create a dataset with 246,161 triples where "Provides" is used to model the relationship between "Service" and "Provider"s providing them, while an instance of "Service" has multiple instances of properties "Capability", "Function" and "Type", and an instance of "Provider" contains multiple instances of properties "Feature" and "Type". There are totally 10,000 "Service" instances and 3,628 "Provider" instances with 10 "Type" instances and 5 instances of "Capability", "Function" and "Feature", respectively.

We extract the Artist-Work subset of the DBpedia data with 40,604 artists with five kinds of identities [1], 136,048 works from twelve kinds of areas [2], sixteen types of links [3][4][5], and four kinds of attributes [6]. We build a heterogeneous network where entity vertices represent artists or works, attribute vertices denote entity's attributes, entity edges represent the relationship between entities, attribute edges specify the interaction between entities and attributes.

We use a subset of the DBLP bibliography data with 200,000 highly prolific authors and associated conferences. We build a heterogeneous network where vertices represent authors and conferences, links represent the number of coauthor works or the number of author's works on conference, and two relevant attributes: prolific and primary topic.

8.6.2 Comparison Methods and Evaluation

We compare **ServiceCluster** with two recently developed representative graph clustering algorithms, **BAGC** [21] and **Inc-Cluster** [28], and one baseline clustering algorithm, **W-**

[1]The artist type: Actor, Comedian, ComicsCreator, MusicalArtist, Writer.

[2]The work type: Album, Book, ComicsCharacter, FictionalCharacter, Film, Magazine, Musical, Newspaper, Single, Song, TelevisionEpisode and TelevisionShow.

[3]The link type between artists: associatedMusicalArtist, influenced, influencedBy.

[4]The link type between works: album, basedOn.

[5]The link type between artists and works: artist, author, creator, director, editor, lyrics, musicalArtist, musicBy, producer, starring, writer.

[6]The attribute type: genre, literaryGenre, occupation, type.

Cluster. ServiceCluster is our proposed algorithm which not only incorporates multiple types of entities, attributes and links into a unified distance model but also continuously enhances the clustering quality by simultaneously refining the link weights, the k value and the cluster centroids. Other three algorithms only integrate structural and attribute information to produce a clustering result. BAGC constructs a probabilistic inference model to capture both structural and attribute aspects. Inc-Cluster combines both structural and attribute similarities in the clustering decisions by estimating the importance of attributes. W-Cluster combines structural and attribute similarities with the equal weighting factors.

Evaluation Measures We use three measures to evaluate the quality of clusters $\{C_l\}_{l=1}^{k_i}$ generated by different methods. The definitions of the metrics are given as follows.

$$density(\{C_l\}_{l=1}^{k_i}) = \sum_{j=1}^{k_i} \frac{|\{(v_p, v_q)|v_p, v_q \in C_j, (v_p, v_q) \in E\}|}{|E|} \tag{8.13}$$

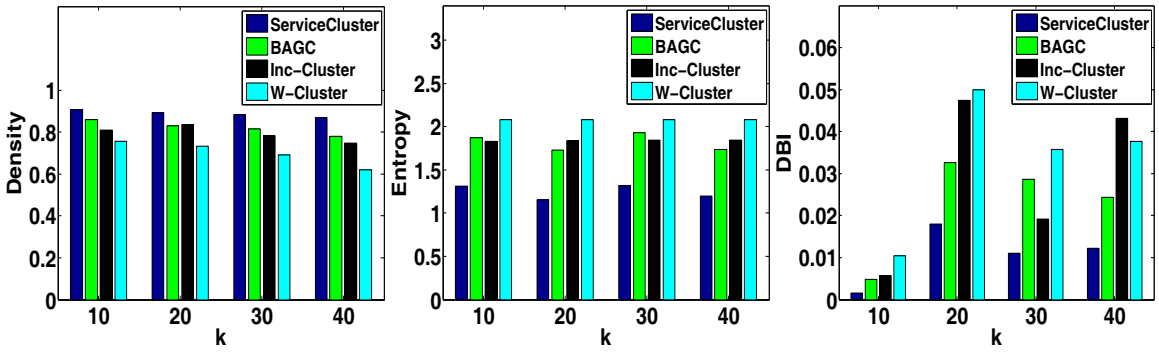
$$entropy(\{C_l\}_{l=1}^{k_i}) = \sum_{p=1}^{M} \frac{\omega_{ip}}{\sum_{q=1}^{M} \omega_{iq}} \sum_{j=1}^{k_i} \frac{|C_j|}{|V_i|} entropy(A_p, C_j) \tag{8.14}$$

where ω_{ip} is the weight between the objective entities V_i and the attribute A_p, $entropy(A_p, C_j) = -\sum_{m=1}^{m_p} p_{pjm} log_2 p_{pjm}$, m_p is the number of A_p's values and p_{pjm} is the percentage of entities in cluster C_j which have m^{th} value on A_p. $entropy(\{C_l\}_{l=1}^{k_i})$ measures the weighted entropy from all attributes over k_i clusters.

Davies-Bouldin Index (DBI) measures the uniqueness of clusters with respect to the unified similarity measure.

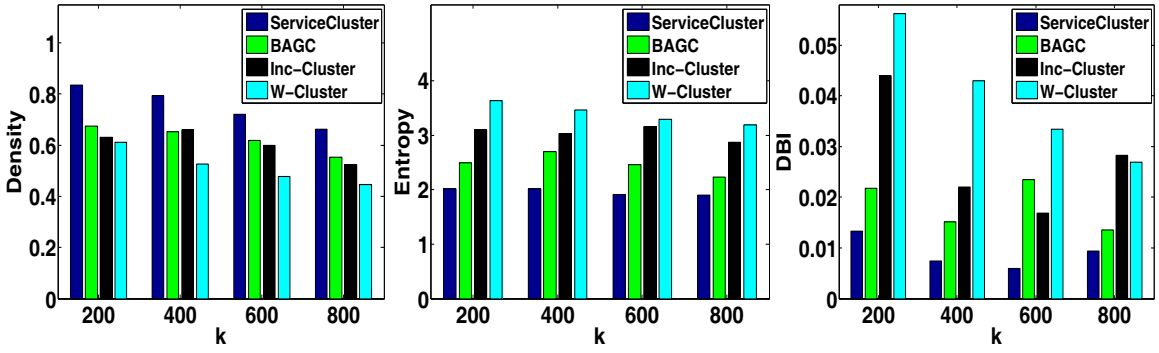
$$DBI(\{C_l\}_{l=1}^{k_i}) = \frac{1}{k_i} \sum_{p=1}^{k_i} max_{q \neq p} \frac{d(c_p, c_q)}{\sigma_p + \sigma_q} \tag{8.15}$$

where c_x is the centroid of C_x, $d(c_p, c_q)$ is the similarity between c_p and c_q, σ_x is the average similarity of entities in C_x to c_x.



(a) density (b) entropy (c) DBI

Figure 8.2: Cluster Quality on BSBM 10,000 Services

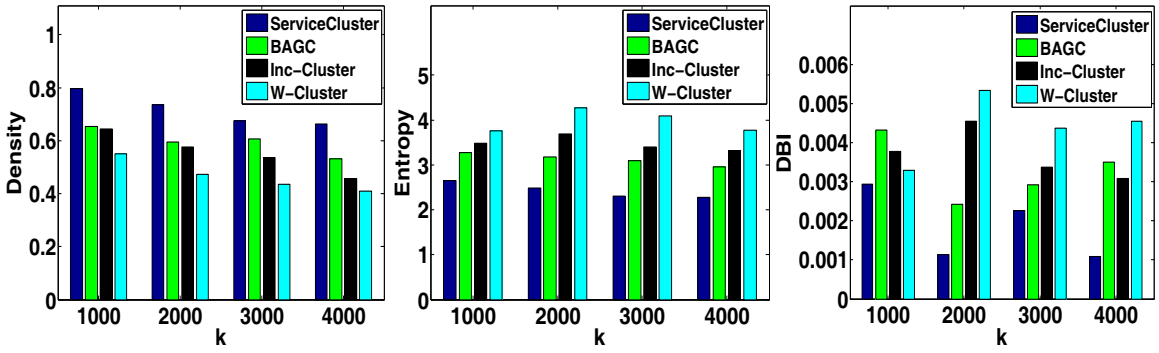


(a) density (b) entropy (c) DBI

Figure 8.3: Cluster Quality on DBpedia 40,604 Artists

8.6.3 Cluster Quality Evaluation

Figures 8.2-8.4 show the quality comparison on three datasets with different k values. To make a fair comparison among different methods, we use a version of ServiceCluster with fixed k to compare other methods with the same k. Figure 8.2 (a) shows the density comparison on BSBM 10,000 Services by varying the number of clusters $k = 10, 20, 30, 40$. The density values by ServiceCluster, BAGC and Inc-Cluster remain 0.74 or higher even when k is increasing. This demonstrates that these methods can find densely connected components. However, ServiceCluster achieves a much higher density than other methods since it not only utilizes the link information between the objective entities but also integrates the interaction among the objective entities, other entities and relevant properties to improve the clustering quality. The density values by W-Cluster is relatively lower, in the



(a) density (b) entropy (c) DBI

Figure 8.4: Cluster Quality on DBLP 200,000 Authors

range of 0.62-0.75 with increasing k, showing that the generated clusters have a very loose intra-cluster structure.

Figure 8.3 (b) shows the entropy comparison on BSBM 10,000 Services with $k = 10, 20, 30, 40$. ServiceCluster has the lowest entropy, while other three algorithms have a much higher entropy, since it not only considers the interaction between the objective entities and their associated attributes but also the interplay between the relevant entities and their attributes. Different from other iterative local maximization approaches, ServiceCluster generates the near global optimal weight assignment for each kind of links by directly solving the maximization problem of clustering objective.

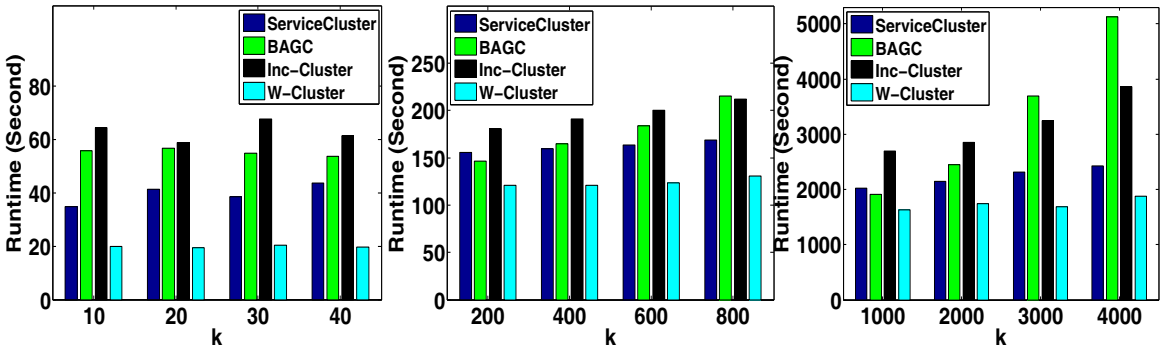
Figure 8.3 (c) shows the DBI comparison on BSBM 10,000 Services with different k values. ServiceCluster has the lowest DBI of 0.003-0.018, while other methods have a much higher DBI than ServiceCluster. This demonstrates that ServiceCluster can obtain both high intra-cluster similarity and low inter-cluster similarity. This is because ServiceCluster incorporates multiple types of entities, attributes, and links with the near global optimal weight assignment. It fully utilizes the connection among the objective entities and other relevant entities, and the interaction between the entities and their attributes such that the generated clusters have not only similar collaborative patterns but also similar interplay patterns with relevant entities and associated attributes.

Similar trends are observed for the quality comparison on other two datasets in Figures

8.3 and 8.4: ServiceCluster achieves the highest density values (>0.66) but the lowest entropy around 1.89-2.64, which is obviously better than the other methods (>2.24). As k increases, the entropy by ServiceCluster remains stable, while the density of ServiceCluster decreases. In addition, ServiceCluster achieves the lowest DBI (0.001-0.013) among different methods.

8.6.4 Clustering Efficiency Evaluation

Figures 8.5 (a) and (b) show the clustering time on three real datasets respectively. ServiceCluster outperforms BAGC and Inc-Cluster in all experiments. We make the following observations on the runtime costs of different methods. First, W-Cluster with a fixed weight assignment is obviously better than other methods since it computes the random walk distance and does graph clustering only once. After each clustering iteration, ServiceCluster and Inc-Cluster need to incrementally update the random walk distance matrix. The cost of incremental distance update is relatively trivial in comparison with recalculating the matrix from scratch. Second, ServiceCluster is much faster than BAGC and Inc-Cluster since it figures out the near global optimal weight assignment by using the successive convex approximation method. Other two local maximization methods by using iterative probabilistic influence or majority vote strategy often converge to a local maximum, even converge to a local minimum or cycle between two points such that they need more iterations to terminate the clustering process. Third, BAGC is much slower than and Inc-Cluster when facing large k values since it is hypersensitive to k. Although BAGC does not need to repeatedly compute the distance matrix, it needs to iteratively update a clustering membership matrix with the size of $n \times k$ and lots of temporary matrices or interim variables such as $\widetilde{\xi}$, $\widetilde{\gamma}$, $\widetilde{\mu}$, $\widetilde{\nu}$ and $\widetilde{\beta}$. As a result, its computational cost is proportional to n^2k^2 such that it may not work well when facing large k values.

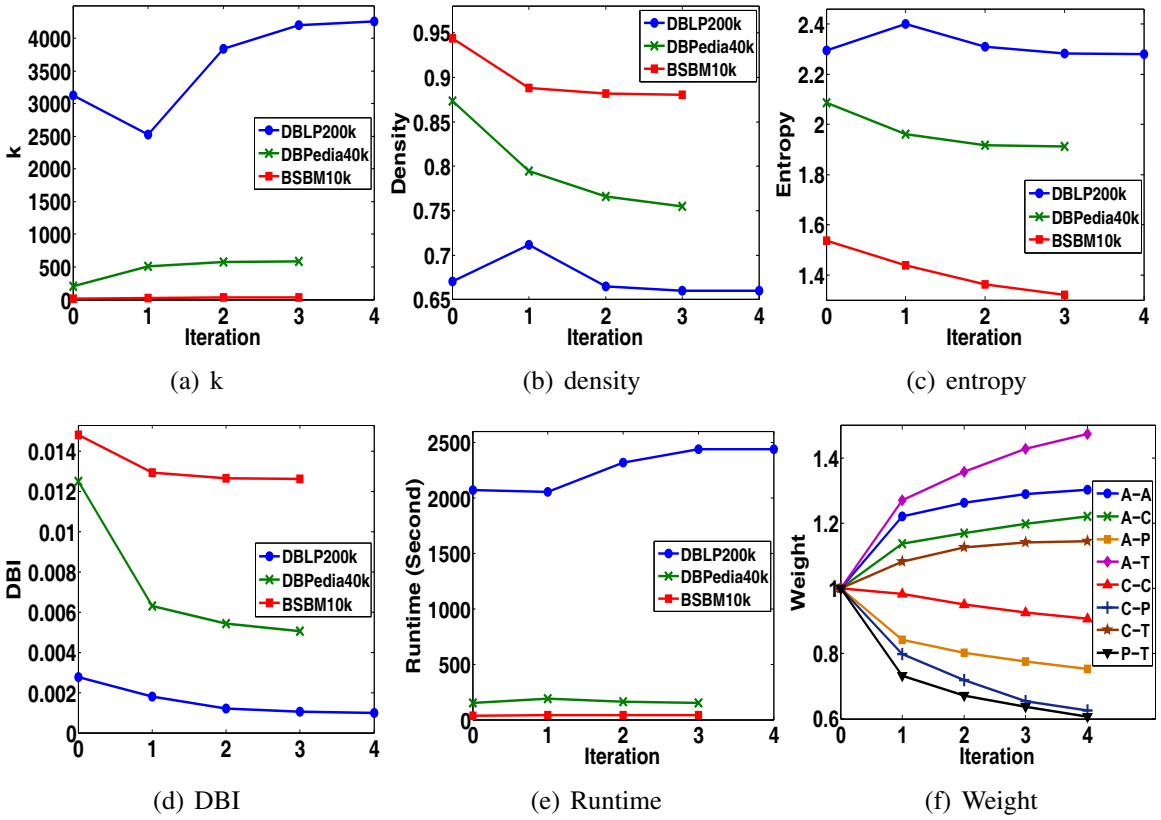


(a) BSBM 10,000 (b) DBpedia 40,604 (c) DBLP 200,000

Figure 8.5: Clustering Efficiency

8.6.5 Clustering Convergence

Figures 8.6 (a)-(f) show the tendency of clustering convergence of ServiceCluster along with the dynamic update k at each iteration on three datasets respectively. Figure 8.6 (a) shows the convergence trend of k on different datasets. The k value converges very quickly, usually in four to five iterations. This demonstrates the efficiency of the algorithm. Figures 8.6 (b) and (c) show how the clustering quality progresses along with the dynamic update k. We know that both density and entropy may decrease with increasing k. Their decreasing trends are highly correlated to some measure of diffuseness such as average cluster radius (or diameter). There may exist a critical point in the plane with k as x-axis and average cluster radius as y-axis, i.e., the average cluster radius decreases quickly as soon as k falls below the critical point but drops slowly as long as k remains at or above the critical point. When the k value arrives at or above the critical point in enough clustering iterations, both density and entropy finally converge to stable values. Figure 8.6 (d) shows the convergence tendency of DBI along with the k values. A low DBI value identifies a clustering with high intra-cluster similarity and low inter-cluster similarity. ServiceCluster exhibits superior performance on all datasets in terms of the DBI measure. The convergence tendency of DBI is consistent with the convergence trend of k on each dataset. Figures 8.6 (e) shows the curve of running time keeps relatively stable when the k value arrives at or above the critical point. Figure 8.6 (f) shows the trend of weight updates on DBLP 200,000



(a) k (b) density (c) entropy

(d) DBI (e) Runtime (f) Weight

Figure 8.6: Clustering Convergence on Different Datasets

Authors along with the dynamic update k where letters "A" and "C" represent two types of entities: author and conference respectively, and letters "P" and "T" denote two kinds of associated attributes: prolific and primary topic respectively.

8.7 Conclusion

We have presented ServiceCluster, a novel heterogeneous service network clustering framework. First, we integrate multiple types of entities, attributes and links with different semantics into a unified random walk distance model. Second, we design a DSD method to naturally produce initial k and initial centroids simultaneously. Third, a dynamic learning approach is proposed to refine the link weights, the k value and the cluster centroids to constantly improve the clustering quality.

CHAPTER 9
CONCLUSIONS AND PROPOSED WORK

Graph as an expressive data structure is popularly used to model structural relationship between objects in many application domains, such as social networks, web graphs, RDF graphs, sensor networks, protein interaction networks. These graphs typically consist of billions of vertices and edges. Effective and efficient analytics on such huge graphs is widely recognized as a challenging big data research problem. To address this challenge, this dissertation is dedicated to the development of algorithms, architectures and optimization techniques for effective and scalable big data analytics. In this chapter, we first summarize the main contributions of this dissertation and then discuss our future research directions.

9.1 Summary

In summary, this dissertation makes three unique contributions.

First, this dissertation proposes a suite of novel graph mining algorithms to analyze and mine large-scale real heterogeneous information networks. The proposed algorithmic approaches enable new ways to dive into the topology and content of big graphs to derive new insights about how heterogeneous entities interact with each other and influence the effectiveness and efficiency of graph clustering, graph classification and graph ranking. The proposed graph mining algorithms have been applied to massive real-world graph datasets, such as Amazon product co-purchasing network, DBLP bibliography data, Last.fm social network, IMDb movie database and Yelp's academic dataset, demonstrating the effectiveness and efficiency of our mining approaches against the state-of-the-art methods.

Second, it introduces a scalable graph processing frmework to speed up the execution of a wide range of real-world graph applications, including graph traversal algorithms, numerical matrix computations, and advanced data mining and machine learning algorithms.

The proposed systemic methods enable computer hardware resource-aware graph partitioning such that parallel graph processing workloads can be well balanced in the presence of highly irregular graph structures and the mismatch of graph access and computation workloads. The proposed multi-level hierarchical graph parallel abstraction can support resource-adaptive selection of the right level of graph parallel abstraction for partitioning, storing and accessing large graphs.

Last but not the least, this dissertation establishes a number of domain specific graph analytics frameworks to perform deep learning and derive new insights in enterprise storage systems and web service libraries. The multifaceted approach of bringing together optimization techniques from various research areas provides a new way to further improve enterprise system design and implementation, and domain specific data management and analytics. In addition, the research results offer new opportunities for product or service providers to deploy their products or services more efficiently and for customers to meet their increasing computational requirements on large-scale data intensive analysis with guaranteed low latency.

9.2 Open Issues and Future Research

The continued growth of ubiquity and complexity of real-world graphs, and the advent of novel graph applications bring many interesting challenges to the research of big data science. I am interested in integrating theoretical techniques with system principles, from multi-disciplinary research areas, such as data mining, databases, machine learning, parallel and distributed computing, cloud computing, security and privacy, and software engineering, to develop effective and scalable big graph computing frameworks for better understanding of large graphs and their underlying processes. Two challenging and promising areas that I am particularly interested to pursue are below.

9.2.1 Social Recommender System Powered by Graph Mining

Recommender systems deal with information overload by suggesting to users the items that are potentially of their interests. Collaborative Filtering (CF) based recommendation techniques have been widely adopted by many online vendors, such as Amazon and Netflix, to promote their marketing efforts. A crucial challenge for building an effective recommender system is the cold start problem: how to provide accurate recommendations to new users or users with extremely few rating records? With the advent of social information along with rating datasets, recent efforts on social recommender system attempt to integrate rating information with additional social information to handle the cold start issue and improve the prediction accuracy. However, these research efforts usually treat social and rating aspects with the same importance or balance them with manual weight setup. In addition, the integration of multiple information sources also introduces additional noisy data at the same time. In fact, a recent study [160] reports that the additional social information can not always result in substantial performance gains of prediction. To handle this challenge, we aim to utilize the analytics techniques of heterogeneous information networks to improves the quality of recommendation in the presence of data sparsity by learning the latent relationships between different types of entities interconnected through heterogeneous types of links. I am confident that combining the techniques from different research fields is promising in significantly boosting the recommendation quality in the presence of data sparsity.

9.2.2 Privacy Preserving of Social Networks Enhanced by Graph Mining

As more and more rich social media and popular online social networking sites are available, privacy preserving publishing of social network data has become a fundamental problem of the modern information infrastructure. People often unwill to provide their personal information if they knew that the privacy of their data could be compromised. The popular privacy preserving research focuses on the question whether tasks can be run over sani-

tized data. Recently, as increasing volumes of personal and sensitive data are collected and archived by social networks, privacy preserving in publishing social network data becomes an important concern. Unfortunately, most of existing privacy preserving techniques can deal with relational data only, and can not be applied to social network data. Thus, we aim to explore the opportunities of preserving privacy in social network data.

The state-of-the-art anonymization methods on social networks can be categorized into three main categories: K-anonymity based privacy preservation via edge modification, probabilistic privacy preservation via edge randomization, and privacy preservation via generalization. However, the ongoing privacy preserving social network analysis mainly focuses on unweighted social networks. Recently, more and more weighted social network data has been made publicly available, such as DBLP collaboration network, Twitter retweet network, and business transaction network, in which the network edges as well as the corresponding weights are considered to be private. However, there are very few existing efforts to study the anonymization of graphs where edge weights are considered sensitive. In fact, edge weights in many real-world information networks are sensitive, e.g., the number of coauthor publications in DBLP or the rating score in IMDb. On the other hand, edge-weight anonymization is still important even if the vertex identities are anonymized. For instance, if the adversary has prior access to the graph, then an attack can be devised for re-identification of vertices in the anonymized graph. In two recent works [161, 162], the authors studied the anonymization of sensitive edge weights in social networks. However, they did not employ any addition, deletion or generalization techniques to vertices or edges, i.e., only adjusted the weights of each links and kept graph structure unchanged such that there still exists a higher risk for leaking personal information and correlations between vertices.

REFERENCES

[1] D. Kempe, J. Kleinberg, and E. Tardos, "Maximizing the spread of influence through a social network," in *Proc. 2003 ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD'03)*, Washington, DC, 2003, pp. 137–146.

[2] P. Domingos and M. Richardson, "Mining the network value of customers," in *Proc. 2001 ACM SIGKDD Int. Conf. Knowledge Discovery in Databases (KDD'01)*, San Francisco, CA, 2001, pp. 57–66.

[3] H. Ma, H. Yang, M. R. Lyu, and I. King, "Mining social networks using heat diffusion processes for marketing candidates selection," in *Proc. 2008 Int. ACM Conference on Information and Knowledge Management (CIKM'08)*, Napa Valley, CA, 2008, pp. 233–242.

[4] M. Roth, A. Ben-David, D. Deutscher, G. Flysher, I. Horn, A. Leichtberg, N. Leiser, Y. Matias, and R. Merom, "Suggesting friends using the implicit social graph," in *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'10)*, Washington, DC, 2010, pp. 233–242.

[5] S. A. Myers, C. Zhu, and J. Leskovec, "Information diffusion and external influence in networks," in *Proc. 2012 ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD'12)*, Beijing, China, 2012, pp. 33–41.

[6] B. Taskar, E. Segal, and D. Koller, "Probabilistic classification and clustering in relational data," in *Proc. 2001 Int. Joint Conf. Artificial Intelligence (IJCAI'01)*, Seattle, WA: International Joint Conferences on Artificial Intelligence Organization, 2001, pp. 870–878.

[7] D. Cai, Z. Shao, X. He, X. Yan, and J. Han, "Community mining from multi-relational networks," in *Proceedings of the 9th European conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'05)*, Porto, Portugal, 2005, pp. 445–452.

[8] T. Yang, R. Jin, Y. Chi, and S. Zhu, "Combining link and content for community detection: a discriminative approach," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD'09)*, Paris, France, 2009, pp. 927–936.

[9] M. Ji, J. Han, and M. Danilevsky, "Ranking-based classification of heterogeneous information networks," in *Proc. 2011 ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD'11)*, San Diego, CA, 2011, pp. 1298–1306.

[10] X. Yu, Y. Sun, P. Zhao, and J. Han, "Query-driven discovery of semantically similar substructures in heterogeneous networks," in *Proc. 2012 ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD'12)*, Beijing, China, 2012, pp. 1500–1503.

[11] J. Shi and J. Malik, "Normalized cuts and image segmentation," in *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, 2000, pp. 888–905.

[12] M. E. J. Newman and M. Girvan, "Finding and evaluating community structure in networks," in *Phys. Rev. E 69, 026113*, College Park, MD: American Physical Society, 2004.

[13] X. Xu, N. Yuruk, Z. Feng, and T. A. J. Schweiger, "Scan: a structural clustering algorithm for networks," in *Proc. 2007 Int. Conf. Knowledge Discovery and Data Mining (KDD'07)*, San Jose, CA, 2007, pp. 824–833.

[14] V. Satuluri and S. Parthasarathy, "Scalable graph clustering using stochastic flows: applications to community discovery," in *Proc. 2009 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'09)*, Paris, France, 2009, pp. 737–746.

[15] K. Macropol and A. Singh, "Scalable discovery of best clusters on large graphs," in *Proceedings of the 36th International Conference on Very Large Data Bases (VLDB'10) / PVLDB 3(1)*, Singapore: VLDB Endowment Inc., 2010, pp. 693–702.

[16] Y. Tian, R. A. Hankins, and J. M. Patel, "Efficient aggregation for graph summarization," in *Proc. 2008 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'08)*, Vancouver, Canada, 2008, pp. 567–580.

[17] N. Zhang, Y. Tian, and J. M. Patel, "Discovery-driven graph summarization," in *Proceedings of the 26th International Conference on Data Engineering (ICDE'10)*, Long Beach, CA, 2010, pp. 880–891.

[18] E. C. Kenley and Y.-R. Cho, "Entropy-based graph clustering: application to biological and social networks," in *Proceedings of the 11th International Conference on Data Mining (ICDM'11)*, Vancouver, Canada, 2011, pp. 1116–1121.

[19] M. Shiga, I. Takigawa, and H. Mamitsuka, "A spectral clustering approach to optimally combining numericalvectors with a modular network," in *Proc. 2007 ACM SIGKDD Int. Conf. Knowledge Discovery in Databases (KDD'07)*, San Jose, CA, 2007, pp. 647–656.

[20] Y. Zhou, H. Cheng, and J. X. Yu, "Graph clustering based on structural/attribute similarities," in *Proceedings of the 35th International Conference on Very Large Data Bases (VLDB'09)*, Lyon, France, 2009, pp. 718–729.

[21] Z. Xu, Y. Ke, Y. Wang, H. Cheng, and J. Cheng, "A model-based approach to attributed graph clustering," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data (SIGMOD'12)*, Scottsdale, Arizona, 2012, pp. 505–516.

[22] Y. Sun, B. Norick, J. Han, X. Yan, P. S. Yu, and X. Yu, "Integrating meta-path selection with user-guided object clustering in heterogeneous information networks," in *Proc. 2012 ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD'12)*, Beijing, China, 2012, pp. 1348–1356.

[23] Y. Sun, C. C. Aggarwal, and J. Han, "Relation strength-aware clustering of heterogeneous information networks with incomplete attributes," *Proceedings of the VLDB Endowment (PVLDB)*, vol. 5, no. 5, pp. 394–405, 2012.

[24] Y. Sun, J. Han, P. Zhao, Z. Yin, H. Cheng, and T. Wu, "Rankclus: integrating clustering with ranking for heterogenous information network analysis," in *Proc. 2009 Int. Conf. Extending Database Technology (EDBT'09)*, Saint Petersburg, Russia: EDBT Association, 2009, pp. 565–576.

[25] L. Kaufmann and P. Rousseeuw, "Clustering by means of medoids," Y. Dodge, Ed., pp. 405–416, 1987.

[26] R. T. Rockafellar, *Convex Analysis*. Princeton, NJ: Princeton University Press, 1997.

[27] L. Botton and Y. Bengio, "Convergence properties of the k-means algorithms," in *Advances in Neural Information Processing Systems 7 (NIPS'94)*, Denver, CO, 1994, pp. 585–592.

[28] Y. Zhou, H. Cheng, and J. X. Yu, "Clustering large attributed graphs: an efficient incremental approach," in *Proceedings of the 10th IEEE International Conference on Data Mining (ICDM'10)*, Sydney, Australia, 2010, pp. 689–698.

[29] S. Godbole and S. Sarawagi, "Discriminative methods for multi-labeled classification," in *Proc. 2004 Pacific-Asia Conf. Knowledge Discovery and Data Mining (PAKDD'04)*, Sydney, Australia, 2004.

[30] G. Chen, Y. Song, F. Wang, and C. Zhang, "Semi-supervised multi-label learning by solving a sylvester equation," in *Proc. 2008 SIAM Int. Conf. on Data Mining (SDM'08)*, Atlanta, GA, 2008.

[31] J. Read, B. Pfahringer, and G. Holmes, "Multi-label classification using ensembles of pruned sets," in *Proc. 2008 Int. Conf. Data Mining (ICDM'08)*, Pisa, Italy, 2008, pp. 995–1000.

[32] W. Cheng and E. Hullermeier, "Combining instance-based learning and logistic regression for multilabel classification," *Machine Learning*, vol. 76, no. 2-3, pp. 211–225, 2009.

[33] K. Dembczynski, W. Cheng, and E. Hullermeier, "Bayes optimal multilabel classification via probabilistic classifier chains," in *Proc. 2010 Int. Conf. Machine Learning (ICML'10)*, Haifa, Israel, 2010.

[34] M.-L. Zhang and K. Zhang, "Multi-label learning by exploiting label dependency," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD'10)*, Washington, DC, 2010, pp. 999–1008.

[35] Y. Guo and S. Gu, "Multi-label classification using conditional dependency networks," in *Proc. 2011 Int. Joint Conf. Artificial Intelligence (IJCAI'11)*, Barcelona, Spain, 2011, pp. 1300–1305.

[36] L. Sun, S. Ji, and J. Ye, "Hypergraph spectral learning for multi-label classification," in *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD'08)*, Las Vegas, NV, 2008, pp. 668–676.

[37] L. Tang and H. Liu, "Scalable learning of collective behavior based on sparse social dimensions," in *Proc. 2009 Int. Conf. Information and Knowledge Management (CIKM'09)*, Hong Kong, China, 2009, pp. 1107–1116.

[38] S. Peters, Y. Jacob, L. Denoyer, and P. Gallinari, "Iterative multi-label multi-relational classification algorithm for complex social networks," *Social Network Analysis and Mining*, vol. 2, no. 1, pp. 17–29, 2012.

[39] X. Kong, B. Cao, and P. S. Yu, "Multi-label classification by mining label and instance correlations from heterogeneous information networks," in *Proceedings of the 19th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD'13)*, Chicago, IL, 2013, pp. 614–622.

[40] X. Wang and G. Sukthankar, "Multi-label relational neighbor classification using social context features," in *Proceedings of the 19th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD'13)*, Chicago, IL, 2013, pp. 464–472.

[41] B. Wang, Z. Tu, and J. K. Tsotsos, "Dynamic label propagation for semi-supervised multi-class multi-label classification," in *Proc. Int. Conf. Computer Vision (ICCV'13)*, 2013.

[42] Q. Lu and L. Getoor, "Link-based classification," in *Proc. 2003 Int. Conf. Machine Learning (ICML'03)*, Washington, DC, 2003, pp. 496–503.

[43] S. A. Macskassy and F. Provost, "A simple relational classifier," in *Proceedings of the Second Workshop on Multi-Relational Data Mining (MRDM'03)*, Washington, DC, 2003, pp. 64–76.

[44] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Scholkopf, "Learning with local and global consistency," in *Advances in Neural Information Processing Systems 16 (NIPS'03)*, Vancouver and Whistler, Canada, 2003.

[45] S. A. Macskassy and F. Provost, "Classification in networked data: a toolkit and a univariate case study," *Journal of Machine Learning Research*, vol. 8, pp. 935–983, 2007.

[46] J. Neville and D. Jensen, "Relational dependency networks," *Journal of Machine Learning Research*, vol. 8, pp. 653–692, 2007.

[47] C. C. Aggarwal and N. Li, "On node classification in dynamic content-based networks," in *Proc. 2011 SIAM Int. Conf. on Data Mining (SDM'11)*, Mesa, AZ, 2011, pp. 355–366.

[48] S. Bhagat, G. Cormode, and S. Muthukrishnan, "Node classification in social networks," C. C. Aggarwal, Ed., pp. 115–148, 2011.

[49] X. Kong, P. S. Yu, Y. Ding, and D. J. Wild, "Meta path-based collective classification in heterogeneous information networks," in *Proc. 2012 Int. Conf. Information and Knowledge Management (CIKM'12)*, Maui, HI, 2012, pp. 1567–1571.

[50] Y. Sun, Y. Yu, and J. Han, "Ranking-based clustering of heterogeneous information networks with star network schema," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD'09)*, Paris, France, 2009, pp. 797–806.

[51] Y. Zhou and L. Liu, "Social influence based clustering of heterogeneous information networks," in *Proceedings of the 19th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD'13)*, Chicago, IL, 2013, pp. 338–346.

[52] T. Chakraborty, S. Sikdar, V. Tammana, N. Ganguly, and A. Mukherjee, "Computer science fields as ground-truth communities: their impact, rise and fall," in *Proceedings of the IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM'13)*, Niagara Falls, Canada, 2013, pp. 426–433.

[53] R.-E. Fan and C.-J. Lin, "A study on threshold selection for multi-label classification," in *Tech Report*, National Taiwan University, 2007.

[54] X. Zhang, Q. Yuan, S. Zhao, W. Fan, W. Zheng, and Z. Wang, "Multi-label classification without the multi-label cost," in *Proc. 2010 SIAM Int. Conf. on Data Mining (SDM'10)*, Columbus, OH, 2010.

[55] Y. Sun, J. Han, X. Yan, P. S. Yu, and T. Wu, "Pathsim: meta path-based top-k similarity search in heterogeneous information networks," *Proceedings of the VLDB Endowment (PVLDB)*, vol. 4, no. 11, pp. 992–1003, 2011.

[56] X. Yu, Y. Sun, B. Norick, T. Mao, and J. Han, "User guided entity similarity search using meta-path selection in heterogeneous information networks," in *Proc. 2012 Int. Conf. Information and Knowledge Management (CIKM'12)*, Maui, HI, 2012, pp. 2025–2029.

[57] J. Zhang, P. S. Yu, and Z.-H. Zhou, "Meta-path based multi-network collective link prediction," in *Proceedings of the 20th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD'14)*, New York, NY, 2014, pp. 1286–1295.

[58] X. Ren, J. Liu, X. Yu, U. Khandelwal, Q. Gu, L. Wang, and J. Han, "Cluscite: effective citation recommendation by information network-based clustering," in *Proceedings of the 20th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD'14)*, New York, NY, 2014, pp. 821–830.

[59] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques* (3rd ed.) Morgan Kaufmann, 2011.

[60] H. Cheng, Y. Zhou, and J. X. Yu, "Clustering large attributed graphs: a balance between structural and attribute similarities," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 2nd ser., vol. 5, pp. 1–33, 2011.

[61] Y. Zhou and L. Liu, "Clustering analysis in large graphs with rich attributes," in *Data Mining: Foundations and Intelligent Paradigms: Volume 1: Clustering, Association and Classification*, D. E. Holmes and L. C. Jain, Eds., Springer, 2011.

[62] H. Cheng, Y. Zhou, X. Huang, and J. X. Yu, "Clustering large attributed information networks: an efficient incremental computing approach," *Data Mining and Knowledge Discovery (DMKD)*, 3rd ser., vol. 25, pp. 450–477, 2012.

[63] W. Cheng, X. Zhang, Z. Guo, Y. Wu, P. Sullivan, and W. Wang, "Flexible and robust co-regularized multi-domain graph clustering," in *Proceedings of the 19th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD'13)*, Chicago, IL, 2013, pp. 320–328.

[64] B. Perozzi, L. Akoglu, P. I. Sanchez, and E. Muller, "Focused clustering and outlier detection in large attributed graphs," in *Proceedings of the 20th ACM SIGKDD*

Conference on Knowledge Discovery and Data Mining (KDD'14), New York, NY, 2014, pp. 1346–1355.

[65] Y. Zhou and L. Liu, "Activity-edge centric multi-label classification for mining heterogeneous information networks," in *Proceedings of the 20th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD'14)*, New York, NY, 2014, pp. 1276–1285.

[66] —, "Social influence based clustering and optimization over heterogeneous information networks," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1st ser., vol. 10, pp. 1–53, 2015.

[67] J. C. Bezdek, *Pattern Recognition with Fuzzy Objective Function Algorithms*. Plenum Press, 1981.

[68] M. Roubens, "Pattern classification problems and fuzzy sets," *Fuzzy Sets and Systems*, vol. 1, no. 4, pp. 239–253, 1978.

[69] M.-S. Yang, "A survey of fuzzy clustering," *Mathematical and Computer Modelling*, vol. 18, no. 11, pp. 1–16, 1993.

[70] D. E. Gustafson and W. C. Kessel, "Fuzzy clustering with a fuzzy covariance matrix," in *Proc. 1979 IEEE Int. Conf. on Decision and Control (CDC'79)*, San Diego, CA, 1979, pp. 761–766.

[71] J. C. Bezdek and N. R. Pal, "Some new indexes of cluster validity," *IEEE Trans. Systems, Man, and Cybernetics, Part B: Cybernetics*, 3rd ser., vol. 28, pp. 301–315, 1998.

[72] H. Hassar and A. Bensaid, "Validation of fuzzy and crisp c-partitions," in *Proc. 1999 Int. Conf. of the North American Fuzzy Information Processing Society (NAFIPS'99)*, New York, NY, 1999, pp. 342–346.

[73] Y. Liu, Z. Li, H. Xiong, X. Gao, and J. Wu, "Understanding of internal clustering validation measures," in *Proc. 2010 Int. Conf. on Data Mining (ICDM'10)*, Sydney, Australia, 2010, pp. 911–916.

[74] G. Malewicz, M. H. Austern, A. J. C. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: a system for large-scale graph processing," in *Proceedings of the 2010 international conference on Management of data (SIGMOD'10)*, Indianapolis, IN, 2010, pp. 135–146.

[75] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein, "Distributed graphlab: a framework for machine learning and data mining in the

cloud," *Proceedings of the VLDB Endowment (PVLDB)*, vol. 5, no. 8, pp. 716–727, 2012.

[76] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "Powergraph: distributed graph-parallel computation on natural graphs," in *Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI'12)*, Hollywood, CA, 2012, pp. 17–30.

[77] *Giraph*, http://giraph.apache.org/.

[78] B. Shao, H. Wang, and Y. Li, "Trinity: a distributed graph engine on a memory cloud," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD'13)*, New York, NY, 2013, pp. 505–516.

[79] J. E. Gonzalez, R. S. Xin, A. Dave, D. Crankshaw, M. J. Franklin, and I. Stoica, "Graphx: graph processing in a distributed dataflow framework," in *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI'14)*, Broomfield, CO, 2014, pp. 599–613.

[80] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein, "Graphlab: a new framework for parallel machine learning," in *Proc. 2002 Annual Conf. Uncertainty in Artificial Intelligence (UAI'10)*, Catalina Island, USA, 2010, pp. 340–349.

[81] A. Kyrola, G. Blelloch, and C. Guestrin, "Graphchi: large-scale graph computation on just a pc," in *Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI'12)*, Hollywood, CA, 2012, pp. 31–46.

[82] W.-S. Han, S. Lee, K. Park, J.-H. Lee, M.-S. Kim, J. Kim, and H. Yu, "Turbograph: a fast parallel graph engine handling billion-scale graphs in a single pc," in *Proceedings of the 19th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD'13)*, Chicago, IL, 2013, pp. 77–85.

[83] A. Roy, I. Mihailovic, and W. Zwaenepoel, "X-stream: edge-centric graph processing using streaming partitions," in *Proceedings of the 24th ACM Symposium on Operating Systems Principles (SOSP'13)*, Farmington, PA, 2013, pp. 472–488.

[84] W. Xie, G. Wang, D. Bindel, A. Demers, and J. Gehrke, "Fast iterative graph computation with block updates," *Proceedings of the VLDB Endowment (PVLDB)*, vol. 6, no. 14, pp. 2014–2025, 2013.

[85] U Kang, C. E. Tsourakakis, and C. Faloutsos, "Pegasus: a peta-scale graph mining system - implementation and observations," in *Proc. 2009 Int. Conf. on Data Mining (ICDM'09)*, Miami, FL, 2009, pp. 229–238.

[86] R. Power and J. Li, "Piccolo: building fast, distributed programs with partitioned tables," in *Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI'10)*, Vancouver, BC, 2010, pp. 1–14.

[87] U Kang, H. Tong, J. Sun, C.-Y. Lin, and C. Faloutsos, "Gbase: a scalable and general graph management system," in *Proc. 2011 ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD'11)*, San Diego, CA, 2011, pp. 1091–1099.

[88] A. Buluc and J. R. Gilbert, "The combinatorial blas: design, implementation, and applications," *IJHPCA*, vol. 25, no. 4, pp. 496–509, 2011.

[89] R. Cheng, J. Hong, A. Kyrola, Y. Miao, X. Weng, M. Wu, F. Yang, L. Zhou, F. Zhao, and E. Chen, "Kineograph: taking the pulse of a fast-changing and connected world," in *Proceedings of the 7th ACM european conference on Computer Systems (EuroSys'12)*, Bern, Switzerland, 2012, pp. 85–98.

[90] V. Prabhakaran, M. Wu, X. Weng, F. McSherry, L. Zhou, and M. Haridasan, "Managing large graphs on multi-cores with graph awareness," in *Proceedings of the 2012 USENIX conference on Annual Technical Conference (ATC'12)*, Boston, MA, 2012.

[91] Y. Tian, A. Balmin, S. A. Corsten, S. Tatikonda, and J. McPherson, "From "think like a vertex" to "think like a graph"," *Proceedings of the VLDB Endowment (PVLDB)*, vol. 7, no. 3, pp. 193–204, 2013.

[92] P. Yuan, W. Zhang, C. Xie, H. Jin, L. Liu, and K. Lee, "Fast iterative graph computation: a path centric approach," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'14)*, New Orleans, LA, 2014, pp. 401–412.

[93] Y. Zhou, L. Liu, K. Lee, C. Pu, and Q. Zhang, "Fast iterative graph computation with resource aware graph parallel abstractions," in *Proceedings of the 24th ACM Symposium on High-Performance Parallel and Distributed Computing (HPDC'15)*, Portland, OR, 2015, pp. 179–190.

[94] K. Lee, L. Liu, K. Schwan, C. Pu, Q. Zhang, Y. Zhou, E. Yigitoglu, and P. Yuan, "Scaling iterative graph computations with graphmap," in *Proceedings of the 27th IEEE international conference for High Performance Computing, Networking, Storage and Analysis (SC'15)*, Austin, TX, 2015.

[95] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation (NSDI'12)*, San Jose, CA, 2012, pp. 15–28.

[96] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," in *Proc. 7th Int. World Wide Web Conf. (WWW'98)*, Brisbane, Australia, 1998, pp. 107–117.

[97] R. I. Kondor and J. D. Lafferty, "Diffusion kernels on graphs and other discrete input spaces," in *Proc. 2002 Int. Conf. Machine Learning (ICML'02)*, Sydney, Australia: IMLS, 2002, pp. 315–322.

[98] W. Tang, Z. Lu, and I. S. Dhillon, "Clustering with multiple graphs," in *Proc. 2006 Int. Conf. on Data Mining (ICDM'06)*, Hong Kong, China, 2006.

[99] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina, "The eigentrust algorithm for reputation management in p2p networks," in *Proceedings of the 12th international conference on World Wide Web (WWW'03)*, Budapest, Hungary, 2003, pp. 640–651.

[100] H. Tong, C. Faloutsos, and J.-Y. Pan, "Fast random walk with restart and its applications," in *Proc. 2006 Int. Conf. on Data Mining (ICDM'06)*, Hong Kong, 2006, pp. 613–622.

[101] X. Zhu, Z. Ghahramani, and J. Lafferty, "Semi-supervised learning using gaussian fields and harmonic functions," in *Proc. 2003 Int. Conf. Machine Learning (ICML'03)*, Washington, DC, 2003, pp. 912–919.

[102] D. D. Lee and H. S. Seung, "Algorithms for non-negative matrix factorization," in *Advances in Neural Information Processing Systems 13 (NIPS'00)*, Denver, CO, 2000, pp. 556–562.

[103] Y. Koren, "Factorization meets the neighborhood: a multifaceted collaborative filtering model," in *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD'08)*, Las Vegas, NV, 2008, pp. 426–434.

[104] H. Ma, "An experimental study on implicit social recommendation," in *Proc. 2013 Int. ACM SIGIR Conf. Research and Development in Information Retrieval (SIGIR'13)*, Dublin, Ireland, 2013, pp. 73–82.

[105] Y. Zhou, L. Liu, C.-S. Perng, A. Sailer, I. Silva-Lepe, and Z. Su, "Ranking services by service network structure and service attributes," in *Proceedings of the 20th International Conference on Web Service (ICWS'13)*, Santa Clara, CA, 2013, pp. 26–33.

[106] Y. Webscope, *Yahoo! altavista web page hyperlink connectivity graph, circa 2002*, http://webscope.sandbox.yahoo.com/.

[107] H. Kwak, C. Lee, H. Park, and S. Moon, "What is twitter, a social network or a news media?," in *Proc. 2010 Int. World Wide Web Conf. (WWW'10)*, Raleigh, NC, 2010, pp. 591–600.

[108] M. Gjoka, M. Kurant, C. T. Butts, and A. Markopoulou, "Walking in facebook: a case study of unbiased sampling of osns," in *Proc. 2010 Int. Conf. Computer Communications (INFOCOM'10)*, San Diego, CA, 2010, pp. 2498–2506.

[109] http://www.informatik.uni-trier.de/~ley/db/.

[110] http://www.last.fm/api/.

[111] J. Cohen, P. Cohen, S. G. West, and L. S. Aiken, *Applied Multiple Regression/Correlation Analysis for the Behavioral Sciences, 3rd Edition*. Routledge, 2002.

[112] O. Bretscher, *Linear Algebra With Applications, 3rd Edition*. Prentice Hall, 1995.

[113] F. S. Hillier and G. J. Lieberman, *Introduction to Operations Research (IBM)*. Blacklick, OH: Mcgraw-Hill College, 1995.

[114] Y. Zhou, L. Liu, and D. Buttler, "Integrating vertex-centric clustering with edge-centric clustering for meta path graph analysis," in *Proceedings of the 21st ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD'15)*, Sydney, Australia, 2015, pp. 1563–1572.

[115] X. Zhu and Z. Ghahramani, "Learning from labeled and unlabeled data with label propagation," in *CMU CALD Tech Report*, 2002.

[116] M. Bender, G. Brodal, R. Fagerberg, R. Jacob, and E. Vicari, "Optimal sparse matrix dense vector multiplication in the i/o-model," *Theory of Computing Systems*, vol. 47, no. 4, pp. 934–962, 2010.

[117] D. Chakrabarti, Y. Zhan, and C. Faloutsos, "R-mat: a recursive model for graph mining," in *Proc. 2004 SIAM Int. Conf. on Data Mining (SDM'04)*, Lake Buena Vista, FL, 2004, pp. 442–446.

[118] D. A. Bader and K. Madduri, *Gtgraph: a synthetic graph generator suite*, 2006.

[119] P. Erdos and A. Renyi, "On random graphs i," *Publicationes Mathematicae*, vol. 6, pp. 290–297, 1959.

[120] F. Viger and M. Latapy, "Efficient and simple generation of random simple connected graphs with prescribed degree sequence," in *Proceedings of the 11th annual international conference on Computing and Combinatorics (COCOON'05)*, Kunming, China, 2005, pp. 440–449.

[121] J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, and Z. Ghahramani, "Kronecker graphs: an approach to modeling networks," *Journal of Machine Learning Research*, vol. 11, pp. 985–1042, 2010.

[122] K. Elgazzar, A. E. Hassan, and P. Martin, "Clustering wsdl documents to bootstrap the discovery of web services," in *Proceedings of the 8th International Conference on Web Service (ICWS'10)*, Miami, FL, 2010, pp. 147–154.

[123] H. Xiao, Y. Zou, J. Ng, and L. Nigul, "An approach for context-aware service discovery and recommendation," in *Proceedings of the 8th International Conference on Web Service (ICWS'10)*, Miami, FL, 2010, pp. 163–170.

[124] D. Skoutas, D. Sacharidis, A. Simitsis, and T. Sellis, "Ranking and clustering web services using multi-criteria dominance relationships," *IEEE Transactions on Services Computing (TSC)*, vol. 3, no. 3, pp. 163–177, 2010.

[125] M. Almulla, K. Almatori, and H. Yahyaoui, "A qos-based fuzzy model for ranking real world web services," in *Proceedings of the 9th International Conference on Web Service (ICWS'11)*, Washington, DC, 2011, pp. 203–210.

[126] M. Aznag, M. Quafafou, N. Durand, and Z. Jarir, "Multiple representations of web services: discovery, clustering and recommendation," in *Proceedings of the 9th International Conference on Web Service (ICWS'11)*, Washington, DC, 2011, pp. 748–749.

[127] S. Dasgupta, S. Bhat, and Y. Lee, "Taxonomic clustering and query matching for efficient service discovery," in *Proceedings of the 9th International Conference on Web Service (ICWS'11)*, Washington, DC, 2011, pp. 363–370.

[128] G. Liu, Y. Wang, M. A. Orgun, and H. Liu, "Discovering trust networks for the selection of trustworthy service providers in complex contextual social networks," in *Proceedings of the 19th International Conference on Web Service (ICWS'12)*, Honolulu, HI, 2012, pp. 384–391.

[129] H. Q. Yu, X. Zhao, S. Reiff-Marganiec, and J. Domingue, "Linked context: a linked data approach to personalised service provisioning," in *Proceedings of the 19th International Conference on Web Service (ICWS'12)*, Honolulu, HI, 2012, pp. 376–383.

[130] J. M. Kleinberg, "Authoritative sources in a hyperlinked environment," *J. ACM*, 5th ser., vol. 46, pp. 604–632, 1999.

[131] B. Bahmani, A. Chowdhury, and A. Goel, "Fast incremental and personalized pagerank," *Proceedings of the VLDB Endowment (PVLDB)*, vol. 4, no. 3, pp. 173–184, 2010.

[132] B. Gao, T.-Y. Liu, W. Wei, T. Wang, and H. Li, "Semi-supervised ranking on very large graph with rich metadata," in *Proc. 2011 ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD'11)*, San Diego, CA, 2011, pp. 96–104.

[133] B. Bahmani, R. Kumar, M. Mahdian, and E. Upfal, "Pagerank on an evolving graph," in *Proc. 2012 ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD'12)*, Beijing, China, 2012, pp. 24–32.

[134] G. Jeh and J. Widom, "SimRank: a measure of structural-context similarity," in *Proc. 2002 ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD'02)*, Edmonton, Canada, 2002, pp. 538–543.

[135] Q. Mei, J. Guo, and D. Radev, "Divrank: the interplay of prestige and diversity in information networks," in *Proc. 2010 ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD'10)*, Washington, DC, 2010, pp. 1009–1018.

[136] H. Tong, J. He, Z. Wen, R. Konuru, and C.-Y. Lin, "Diversified ranking on large graphs: an optimization viewpoint," in *Proc. 2011 ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD'11)*, San Diego, CA, 2011, pp. 1028–1036.

[137] Y. Zhou, L. Liu, C. Pu, X. Bao, K. Lee, B. Palanisamy, E. Yigitoglu, and Q. Zhang, "Clustering service networks with entity, attribute and link heterogeneity," in *Proceedings of the 22nd International Conference on Web Service (ICWS'15)*, New York, NY, 2015, pp. 257–264.

[138] S. P. Lloyd, "Least squares quantization in pcm," 2nd ser., vol. 28, pp. 128–137, 1982.

[139] C. Bizer and A. Schultz, "The berlin sparql benchmark," *Int. J. Semantic Web Inf. Syst.*, vol. 5, no. 2, pp. 1–24, 2009.

[140] D. Arthur and S. Vassilvitskii, "K-means++: the advantages of careful seeding," in *Proc. 2007 Annual ACM-SIAM Symp. Discrete Algorithms (SODA'07)*, New Orleans, LA, 2007, pp. 1027–1035.

[141] M. Bhadkamkar, J. Guerra, L. Useche, S. Burnett, J. Liptak, R. Rangaswami, and V. Hristidis, "Borg: block-reorganization for self-optimizing storage systems," in *Proccedings of the 7th Conference on File and Storage Technologies (FAST'09)*, San Francisco, CA, 2009, pp. 183–196.

[142] Y. Zhang, G. Soundararajan, M. W. Storer, L. N. Bairavasundaram, S. Subbiah, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Warming up storage-level caches with bonfire," in *Proceedings of the 11th USENIX Conference on File and Storage Technologies (FAST'13)*, San Jose, CA, 2013.

[143] G. Zhang, L. Chiu, C. Dickey, L. Liu, P. Muench, and S. Seshadri, "Automated lookahead data migration in ssd-enabled multi-tiered storage system," in *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST'10)*, Washington, DC, 2010, pp. 1–6.

[144] P. Pipada, A. Kundu, K. Gopinath, C. Bhattacharyya, S. Susarla, and P. C. Nagesh, "Loadiq: learning to identify workload phases from a live storage trace," in *Proceedings of the 4th USENIX conference on Hot Topics in Storage and File Systems (HotStorage'12)*, Boston, MA, 2012, pp. 3–3.

[145] V. Tarasov, D. Hildebrand, G. Kuenning, and E. Zadok, "Virtual machineworkloads: the case for new benchmarks for nas," in *Proceedings of the 11th USENIX Conference on File and Storage Technologies (FAST'13)*, San Jose, CA, 2013.

[146] A. Gulati, C. Kumar, I. Ahmad, and K. Kumar, "Basil: automated io load balancing across storage devices," in *Proceedings of the 8th USENIX Conference on File and Storage Technologies (FAST'10)*, San Jose, CA, 2010, pp. 13–13.

[147] N. Park, I. Ahmad, and D. J. Lilja, "Romano: autonomous storage management using performance prediction in. multitenant datacenters," in *Proceedings of the Third ACM Symposium on Cloud Computing (SoCC'12)*, San Jose, CA, 2012, 21:1–21:14.

[148] Y. Chen, K. Srinivasan, G. R. Goodson, and R. H. Katz, "Design implications for enterprise storage systems via multi-dimensional trace analysis," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles (SOSP'11)*, Cascais, Portugal, 2011, pp. 43–56.

[149] S. Kavalanekar, B. L. Worthington, Q. Zhang, and V. Sharda, "Characterization of storage workload traces from production windows servers," in *Proceedings of the 2008 IEEE International Symposium on Workload Characterization (IISWC'08)*, Seattle, WA, 2008, pp. 119–128.

[150] A. W. Leung, S. Pasupathy, G. Goodson, and E. L. Miller, "Measurement and analysis of large-scale network file system workloads," in *USENIX 2008 Annual Technical Conference on Annual Technical Conference (ATC'08)*, Boston, MA, 2008, pp. 213–226.

[151] G. Wallace, F. Douglis, H. Qian, P. Shilane, S. Smaldone, M. Chamness, and W. Hsu, "Characteristics of backup workloads in production systems," in *Proceedings of the 10th USENIX Conference on File and Storage Technologies (FAST'12)*, San Jose, CA, 2012, pp. 4–4.

[152] H. Hwang, V. Hristidis, and Y. Papakonstantinou, "Objectrank: a system for authority-based search on databases," in *Proc. 2006 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'06)*, Chicago, IL, 2006, pp. 796–798.

[153] S. Byan, J. Lentini, A. Madan, L. Pabon, M. Condict, J. Kimmel, S. Kleiman, C. Small, and M. Storer, "Mercury: host-side flash caching for the data center," in *Proceedings of the 2012 IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST'12)*, Monterey, CA, 2012, pp. 1–12.

[154] J. Guerra, H. Pucha, J. Glider, W. Belluomini, and R. Rangaswami, "Cost effective storage using extent based dynamic tiering," in *Proceedings of the 9th USENIX Conference on File and Stroage Technologies (FAST'11)*, San Jose, CA, 2011, pp. 20–20.

[155] M. Aznag, M. Quafafou, and Z. Jarir, "Leveraging formal concept analysis with topic correlation for service clustering and discovery," in *Proceedings of the 21st International Conference on Web Service (ICWS'14)*, Anchorage, AK, 2014, pp. 153–160.

[156] Z. Su, L. Liu, M. Li, X. Fan, and Y. Zhou, "Servicetrust: trust management in service provision networks," in *Proceedings of the 10th IEEE International Conference on Services Computing (SCC'13)*, Santa Clara, CA, 2013, pp. 272–279.

[157] B. T. G. S. Kumara, I. Paik, and W. Chen, "Web-service clustering with a hybrid of ontology learning and information-retrieval-based term similarity," in *Proceedings of the 20th International Conference on Web Service (ICWS'13)*, Santa Clara, CA, 2013, pp. 340–347.

[158] Z. Su, L. Liu, M. Li, X. Fan, and Y. Zhou, "Reliable and resilient trust management in distributed service provision networks," *ACM Transactions on the Web (TWEB)*, 3rd ser., vol. 9, pp. 1–37, 2015.

[159] G. Hamerly and C. Elkan, "Learning the k in k-means," in *Advances in Neural Information Processing Systems 17 (NIPS'03)*, Vancouver and Whistler, Canada, 2003.

[160] E. Shmueli, A. Kagian, Y. Koren, and R. Lempel, "Care to comment? recommendations for commenting on news stories," in *Proc. 21st Int. Conf. World Wide Web (WWW'12)*, Lyon, France, 2012, pp. 429–438.

[161] S. Das, O. Egecioglu, and A. E. Abbadi, "Anonymizing edge-weighted social network graphs," in *UCSB Computer Science Technical Report*, 2009.

[162] L. Liu, J. Wang, J. Liu, and J. Zhang, “Privacy preservation in social networks with sensitive edge weights,” in *Proc. 2009 SIAM Int. Conf. on Data Mining (SDM’09)*, Sparks, NV, 2009, pp. 954–965.